Broadview
www.broadview.com.cn

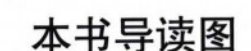
MATLAB
精品丛书

精通MATLAB神经网络

朱凯 王正林 编著

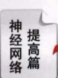
- 实例丰富、高效实用
- 语言简练、通俗易懂
- 内容详实、全面系统



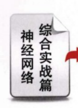




了解MATLAB软件，掌握其运算和绘图两大功能，熟练掌握程序设计以及Simulink仿真是应用MATLAB神经网络的基础。



从应用的角度出发，以讲明基本概念和方法为主，尽量减少繁琐的数学推导，重点介绍实用的网络模型、学习规则和训练方法，以及MATLAB神经网络工具箱的实现，并提供丰富的工程应用实例。



神经网络在最优化、控制、故障诊断、预测等应用领域具有广泛而深入的应用，利用MATLAB神经网络工具箱来解决这些问题，快速而便捷。

本篇通过综合实例，将MATLAB神经网络工具箱与具体应用结合起来，完整、有条理地讲述应用MATLAB神经网络工具箱的具体流程。

附录A

对神经网络工具箱中所有函数的功能、输入/输出参数说明及参数默认值等进行了详细的说明，非常便于读者进行查询和参考。

图书分类：计算机>MATLAB




ISBN 978-7-121-09985-4
9 787121 099854 >
定价：59.00元（含光盘1张）

PHEI
责任编辑：朱沭红
责任美编：李　玲



精通MATLAB神经网络

朱凯　王正林

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书以应用为方向，以实用为目标来讲述MATLAB的神经网络计算，在简要介绍神经网络的各种典型网络以及训练过程之后，重点讲述应用MATLAB神经网络工具箱进行神经网络的设计与应用，并辅以大量的实例及综合实战应用。

本书由MATLAB入门篇、神经网络提高篇和神经网络综合实战篇3篇组成。MATLAB入门篇主要介绍MATLAB软件、基本运算、图形绘制、程序设计和Simulink仿真；神经网络提高篇讲述神经网络的主要内容，包括神经网络工具箱和GUI工具，以及感知器、线性、BP、径向基、自组织、反馈等各种不同的神经网络，讲述各种神经网络的性能分析与直观的图形结果，使读者更加透彻地了解各种神经网络的性能及其优缺点，从而达到理解和应用神经网络的目的。综合实战篇通过综合实例，讲述应用MATLAB来分析、解决具体的神经网络问题，包括神经网络优化、控制、故障诊断和预测等典型应用，还讲述了Simulink神经网络设计及自定义神经网络。

本书语言简练、实例丰富、阐述清晰，可供计算机、信息处理、控制、地球物理、医学、管理等专业的本科生、研究生作为神经网络教材和参考书使用，也可供相应的工程技术人员参考使用。

图书在版编目（CIP）数据

精通MATLAB神经网络 / 朱凯，王正林编著. —北京：电子工业出版社，2010.1
（MATLAB精品丛书）
ISBN 978-7-121-09985-4

I. 精… II. ①朱… ②王… III. 计算机辅助计算－软件包，Matlab－应用－神经网络
IV. TP391.75 TP183

中国版本图书馆CIP数据核字（2009）第218369号

责任编辑：朱沭红
印　　刷：北京智力达印刷有限公司
装　　订：北京中新伟业印刷有限公司
出版发行：电子工业出版社
　　　　　北京市海淀区万寿路173信箱　邮编 100036
开　　本：787×1092　1/16　印张：28.5　字数：640.7千字
印　　次：2010年1月第1次印刷
印　　数：4000册　定价：59.00元（含光盘1张）

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。
质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。
服务热线：（010）88258888。

前　　言

神经网络是一种能适应新环境的系统，它通过对过去经验（信息）的重复学习，而具有分析、预测、推理、分类等功能，是当今能够仿效人类大脑去解决复杂问题的系统，神经网络是人工智能技术领域的重要方向，在自动控制、信号处理、工业控制，以及模式识别等诸多领域获得了广泛的应用。

随着计算机技术的迅猛发展，神经网络技术与计算机技术不断融合，催生了一系列可用于神经网络的软件，如 NeuroSolutions、NeuralSight、MATLAB 等，这些软件的广泛应用，极大地推动了神经网络的发展。

MATLAB 已成为国际公认的最优秀的科技应用软件，具有编程简单、数据可视化功能强、可操作性强等特点，而且具备功能强大、专业函数丰富的神经网络工具箱，是进行神经网络方面工作必备的高效软件工具。

利用 MATLAB 的神经网络工具箱，通过任意连接及合成不同的网络架构以实现类神经网络仿真及专业化应用，而且还具有美观的操作界面，可以快速构建出相应的神经网络，便捷地训练、测试网络。

本书导读

MATLAB入门篇

MATLAB 具有强大的计算功能和数据可视化功能，其中包括 Simulink 仿真工具，是首选的科学计算软件工具。

了解 MATLAB 软件，掌握其运算和绘图两大功能，熟练掌握程序设计以及 Simulink 仿真是应用 MATLAB 神经网络的基础。

神经网络提高篇

从应用神经网络工具箱的角度出发，以讲明基本概念和方法为主，尽量减少烦琐的数学推导，重点介绍实用的网络模型、学习规则和训练方法，以及 MATLAB 神经网络工具箱的实现，并且提供丰富的工程应用实例。

内容全面，重点突出，语言简洁，叙述清晰，实例丰富，读者能快速掌握。

神经网络综合实战篇

神经网络在最优化、控制、故障诊断、预测等应用领域具有广泛而深入的应用，利用 MATLAB 神经网络工具箱来解决这些问题，快速而便捷。

本篇通过综合实例，将 MATLAB 神经网络工具箱与具体应用结合起来，完整、有条理地讲述应用 MATLAB 神经网络工具箱的整个具体流程。

附录A

对神经网络工具箱的所有函数功能、输入/输出参数及参数默认值等进行了详细的说明，非常便于读者进行查询和参考。

通过本附录，读者在进行 MATLAB 神经网络计算时能方便快捷地查询到相关函数的使用方法。

本书读者

本书可作为从事计算机、信息处理、控制、地球物理、医学、管理等专业的本科生，研究生学习神经网络的辅助教材和参考书，也可供相应的工程技术人员参考使用。

作者致谢

本书在编写过程中，尤其是综合实战篇部分，参考、借鉴了一些国内外书籍、论文和文献资料，对此特向各位从事神经网络研究和应用工作的专家老师们表示真诚的敬意和感谢！

最后，感谢父母和朋友们的支持与鼓励，使得本书的创作过程得以坚持下去；感谢朱沭红老师、许艳老师的大力支持和辛勤劳动！

由于作者水平和经验有限，书中错漏之处在所难免，还望得到专家、读者和行内人士的批评指正，我们的邮箱是：wa_2003@126.com。

编著者　2009 年 11 月 18 日



目　录

第一篇　MATLAB 入门篇

第二篇 神经网络提高篇

第三篇 神经网络综合实战篇

Part 1

MATLAB 入门篇



第 1 章 MATLAB 概述

经过 20 余年的补充与完善以及多个版本的升级换代，MATLAB 已发展至 R2009b 版本。MATLAB 是一个包含众多科学、工程计算的庞大系统，是目前世界上最流行的计算软件之一。

1.1 MATLAB 的产生与发展

MATLAB 语言的产生是与数学计算紧密联系在一起的。1980 年，美国新墨西哥州大学计算机系主任 Cleve Moler 在给学生讲授线性代数课程时，发现学生在高级语言编程上花费很多时间，于是着手编写供学生使用的 Fortran 子程序库接口程序，他将这个接口程序取名为 MATLAB（即 Matrix Laboratory 的前三个字母的组合，意为“矩阵实验室”）。这个程序获得了很大的成功，受到学生的广泛欢迎。

20 世纪 80 年代初期，Moler 等一批数学家与软件专家组建了 MathWorks 软件开发公司，继续从事 MATLAB 的研究和开发，1984 年推出了第一个 MATLAB 商业版本，其核心是用 C 语言编写的。而后，它又添加了丰富多彩的图形图像处理、多媒体、符号运算，以及与其他流行软件的接口功能，使得 MATLAB 的功能越来越强大。

MathWorks 公司正式推出 MATLAB 后，于 1992 年推出了具有划时代意义的 MATLAB 4.0 版本，之后陆续推出了几个改进和提高的版本，2004 年 9 月正式推出 MATLAB Release 14，即 MATLAB 7.0，其功能在原有的基础上又有了进一步的改进，2009 年 9 月推出了 R2009b，它是目前 MATLAB 最新的版本。

MATLAB 经过几十年的研究与不断完善，现已成为国际上最为流行的科学计算与工程计算软件工具之一，现在的 MATLAB 已经不仅仅是一个最初的“矩阵实验室”了，它已发展成为一种具有广泛应用前景、全新的计算机高级编程语言，可以说它是“第四代”计算机语言。

自 20 世纪 90 年代以来，美国和欧洲的各大学将 MATLAB 正式列入研究生和本科生的教学计划，MATLAB 软件已成为数值计算、数理统计、信号处理、时间序列分析、动态系统仿真等课程的基本教学工具，成为学生必须掌握的基本软件之一。在研究单位和工业界，MATLAB 也成为工程师们必须掌握的一种工具，被认做进行高效研究与开发的首选软件工具。

1.2 MATLAB 的优势与特点

MATLAB 在学术界和工程界广受欢迎，其主要优势和特点有如下几方面。

● 友好的工作平台和编程环境

MATLAB 由一系列工具组成，其中许多工具采用的是图形用户界面，包括 MATLAB 桌面和命令窗口、历史命令窗口、编辑器和调试器、路径搜索和用于用户浏览帮助、工作空间、文件的浏览器。这些图形化的工具方便用户使用 MATLAB 的函数和文件。

随着 MATLAB 的商业化以及软件本身的不断升级，MATLAB 的用户界面也越来越精致，更加接近 Windows 的标准界面，人机交互性更强，操作更简单。

同时，MATLAB 提供了完整的联机查询、帮助系统，极大地方便了用户的使用。

MATLAB 简单的编程环境提供了比较完备的调试系统，程序不必经过编译就可以直接运行，而且能够及时地报告出现的错误并进行出错原因分析。

● 简单易用的编程语言

MATLAB 语言是一种高级的矩阵语言，它包含控制语句、函数、数据结构、输入和输出，具有面向对象编程的特点。用户可以在命令窗口中将输入语句与执行命令同步，也可以先编写好一个较大的复杂应用程序（M 文件）后再一起运行。

MATLAB 语言是基于流行的 C++语言的，因此语法特征与 C++语言极为相似，而且更加简单，更加符合科技人员对数学表达式的书写格式，更利于非计算机专业的科技人员使用。而且这种语言可移植性好、可拓展性强，这也是 MATLAB 能够深入到科学研究及工程计算各个领域的重要原因。

● 强大的科学计算数据处理能力

MATLAB 是一个包含大量计算算法的集合，其拥有 600 多个工程中要用到的数学运算函数，可以方便地实现用户所需的各种计算功能。

函数所能解决的问题大致包括矩阵运算和线性方程组的求解、微分方程及偏微分方程的组的求解、符号运算、傅里叶变换，数据的统计分析、工程中的优化问题、稀疏矩阵运算、复数的各种运算、三角函数和其他初等数学运算、多维数组操作，以及建模动态仿真等。函数中所使用的算法都是科研和工程计算中的最新研究成果，而且经过了各种优化和容错处理。

在通常情况下，可以用 MATLAB 来代替底层编程语言，如 C 和 C++。在计算要求相同的情况下，使用 MATLAB 的编程工作量会大大减少。

● 出色的图形处理功能

MATLAB 自产生之日起就具有方便的数据可视化功能，能够将向量和矩阵用图形的形式表现出来，并且可以对图形进行标注和打印。

MATLAB 的高层次的作图功能包括二维和三维的可视化、图像处理、动画和表达式作图，可用于科学计算和工程绘图。

MATLAB 对整个图形处理功能进行了很大的改进和完善，使它不仅在一般数据可视化软件都具有的功能（例如二维曲线和三维曲面的绘制和处理等）方面更加完善，而且对于一些其他软件所没有的功能（例如图形的光照处理、色度处理，以及四维数据的表现等），MATLAB 同样表现出色。

对一些特殊的可视化要求，例如图形对话等，MATLAB 也有相应的功能函数，保证了

用户不同层次的要求。MATLAB 还在图形用户界面（GUI）的制作上作了很大的改善，对这方面有特殊要求的用户也可以得到满足。

- 应用广泛的模块集合工具箱

MATLAB 对许多专门的领域都开发了功能强大的模块集和工具箱（Toolbox）。一般来说，它们都是由特定领域的专家开发的，用户可以直接使用工具箱学习、应用和评估不同的方法而不需要自己编写代码。

目前，MATLAB 已经把工具箱延伸到了科学研究和工程应用的诸多领域，如神经网络、小波分析、优化算法、样条拟合、概率统计、偏微分方程求解、信号处理、图像处理、模糊逻辑、金融分析等，都在工具箱家族中有了自己的一席之地。

- 实用的程序接口和发布平台

MATLAB 可以利用 MATLAB 编译器和 C/C++数学库和图形库，将自己的 MATLAB 程序自动转换为独立于 MATLAB 运行的 C 和 C++代码，允许用户编写可以和 MATLAB 进行交互的 C 或 C++语言程序。另外，MATLAB 网页服务程序还容许在 Web 应用中使用自己的 MATLAB 数学和图形程序。

1.3 MATLAB 系统的构成

MATLAB 系统由 MATLAB 开发环境、MATLAB 数学函数库、MATLAB 语言、MATLAB 图形处理系统和 MATLAB 应用程序接口（API）五大部分构成。

- MATLAB 开发环境

这部分是一套方便用户使用 MATLAB 函数和文件的工具集，其中许多工具是友好的、交互式的图形化用户接口。它是一个集成化的工作空间，可以让用户输入、输出数据，并提供了 M 文件的集成编译和调试环境。它包括 MATLAB 桌面、命令窗口、M 文件编辑调试器、代码分析器（Code Analyzer）、查看帮助、工作空间、文件和其他工具的浏览器。

- MATLAB 数学函数库

MATLAB 数学函数库包括了大量的计算算法，从基本运算（如加法、正弦函数等）到复杂算法，如矩阵求逆、矩阵求特征值、贝塞尔函数、快速傅里叶变换等。

- MATLAB 语言

MATLAB 语言是一个高级的基于矩阵/数组的语言，它有程序流控制、函数、数据结构、输入/输出和面向对象编程等特色。用户既可以用它来快速编写简单的程序，也可以用它来编写庞大复杂、重用性高的应用程序。

- MATLAB 图形处理系统

图形处理系统使得 MATLAB 能方便地图形化显示向量和矩阵，而且能对图形添加标注和打印。MATLAB 提供两个层次的绘图操作，一种是对图形句柄进行的底层绘图操作，另一种是建立在底层绘图操作之上的高层绘图操作。

- MATLAB 应用程序接口

MATLAB 应用程序接口是一个使 MATLAB 与 C、Fortran 等其他高级编程语言进行交

互的函数库，该函数库的函数通过调用动态连接库（DLL）实现与 MATLAB 文件的数据交换，其主要功能包括在 MATLAB 中调用 C 和 Fortran 程序，以及在 MATLAB 与其他应用程序间建立客户/服务器关系。

1.4 MATLAB 桌面操作环境

MATLAB 为用户提供了全新的桌面操作环境，了解并熟悉这些桌面操作环境是使用 MATLAB 的基础，下面介绍 MATLAB 的启动、主要功能菜单、命令窗口（Command Window）、工作空间（Workspace）、文件管理和帮助管理等。

1.4.1 MATLAB 启动和退出

以 Windows 操作系统为例，进入 Windows 后，选择“开始”→“程序”→“MATLAB R2008b”，便可以进入如图 1-1 所示的 MATLAB 默认主窗口。如果安装时选择在桌面上生成快捷方式，也可以双击快捷方式直接启动。

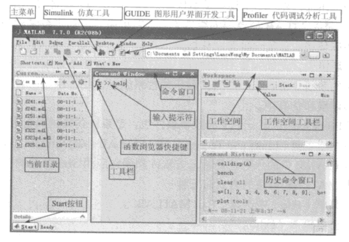


图 1-1 MATLAB 主窗口

MATLAB 主窗口是 MATLAB 的主要工作界面。主窗口除了嵌入一些子窗口外，还包括菜单栏和工具栏。

主窗口的工具栏共提供了 11 个命令按钮。这些命令按钮均有对应的菜单命令，但比菜单命令使用起来更快捷、方便。

单击主窗口左下角的“Start”按钮，会弹出一个菜单，如图 1-2 所示。选择其中的命令可以执行 MATLAB 产品的各种工具，并且可以查阅 MATLAB 包含的各种资源。

从图 1-2 中可以看出，MATLAB 的主要资源有：

- MATLAB 主体：由 MATLAB 的编程集成环境、程序开发工具和与其他软件的扩展接口组成。

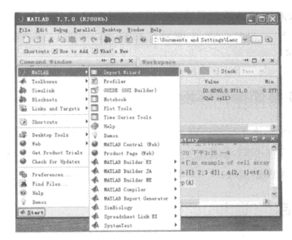


图 1-2 “Start”按钮的弹出菜单

- 工具箱（Toolboxes）：是 MATLAB 函数的子程序库，每一个工具箱都是为某一类学科专业和应用而定制的，主要包括最优化计算、遗传算法、神经网络等方面的应用。
- Simulink：是 MATLAB 最重要的组件之一，它提供一个动态系统建模、仿真和综合分析的集成环境。它是一种可视化仿真工具，是一种基于 MATLAB 的框图设计环境，在该环境中，无须大量书写程序，而只需要通过简单直观的鼠标操作，就可构造出复杂的系统。Simulink 广泛应用于线性系统、非线性系统、数字控制及数字信号处理的建模和仿真中。
- 模块集（Blocksets）：是一个个的数学软件包，是系统仿真的关键部件。
- 自动代码生成工具（Links and Targets）：将 MATLAB 中的 Simulink 程序框图自动转换成嵌入式 ANSI C 的代码，是第三方软件和硬件应用 Simulink 的工具。

常用的退出 MATLAB 系统的方式有三种：

（1）在文件菜单（File）中选择“Exit MATLAB”。

（2）在命令窗口输入“exit”。

（3）用鼠标单击窗口右上角的关闭图标。

1.4.2 MATLAB 主菜单及功能

打开 MATLAB 主窗口后，即弹出其主菜单栏，共包含 File、Edit、Debug、Parallel、Desktop、Window 和 Help 共 7 个菜单项。主菜单栏的各菜单项及其下拉菜单的功能简要介绍如下。

1. File 主菜单

File 主菜单实现有关文件的操作，其下拉菜单项有：

（1）New：用于建立新的.m 文件、图形、模型和图形用户界面。

（2）Open：用于打开 MATLAB 的.m 文件、.fig 文件、.mat 文件、.mdl 文件、.cdr 文

件等，也可通过快捷键“Ctrl+O”来实现此项操作。

（3）Close Command Window：关闭命令窗口。

（4）Import Data：用于从其他文件导入数据，单击此菜单项将弹出对话框，可在其中选择导入文件的路径和位置。

（5）Save Workspace As：用于把工作空间的数据存放到相应的路径文件中。

（6）Set Path：设置工作路径。

（7）Preferences：用于设置命令窗的属性，单击该选项弹出一个属性画面。

（8）Page Setup：用于页面设置。

（9）Print：用于设置打印属性。

（10）Print Selection：用于对选择的文件数据进行打印设置。

（11）Exit MATLAB：退出 MATLAB 桌面操作环境。

2. Edit 主菜单

Edit 主菜单用于命令窗口的编辑操作，其下拉菜单项有：

（1）Undo：用于撤销上一步操作。

（2）Redo：用于重新执行上一步操作。

（3）Cut：用于剪切选中的对象。

（4）Copy：用于复制选中的对象。

（5）Paste：用于粘贴剪贴板上的内容。

（6）Paste to Workspace：用于打开 Import Wizard（输入向导）对话框，将剪贴板上的数据粘贴到 MATLAB 的工作空间中。

（7）Select All：用于全部选择。

（8）Delete：用于删除所选的对象。

（9）Find：用于查找所需选择的对象。

（10）Find Files：用于查找所需文件。

（11）Clear Command Window：用于清除命令窗口区的对象。

（12）Clear Command History：用于清除命令窗口区的历史记录。

（13）Clear Workspace：用于清除工作区的对象。

3. Debug 主菜单

用户可以通过 Debug 主菜单进行程序调试时的各种设置，其下拉菜单项有：

（1）Open M-Files when Debugging：用于调试时打开 M 文件。

（2）Step：用于单步调试程序。

（3）Step In：用于单步调试进入子函数。

（4）Step Out：用于单步调试从子函数中跳出。

（5）Continue：程序执行到下一断点。

（6）Clear Breakpoints in All Files：清除所有打开文件中的断点。

（7）Stop if Errors/Warnings：在程序出错或报警处停止往下执行。

（8）Exit Debug Mode：退出调试模式。

4. Parallel 主菜单

Parallel 主菜单用来进行并行计算方面的设置，其下拉菜单项有：

（1）Select Configuration：选择并行计算的配置类型。

（2）Manage Configuration：对配置进行管理。

（3）Admin Center：打开并行计算的管理中心。

并行计算的设置比较专业，一般不进行设置。

5. Desktop 主菜单

Desktop 主菜单用来设置主窗口中需要打开的窗口，其下拉菜单项有：

（1）Desktop Layout：单击该项后，弹出一个子菜单；用于设置桌面显示方式，其设置选项包括系统默认设置项（Default）、单独命令窗口项（Command Window Only）、命令历史窗口和命令窗口项（History and Command Window）、全部标签项显示（All Tabbed）。

（2）Save Layout：保存选定的桌面显示方式设置。

（3）Organize Layouts：管理保存的桌面显示方式设置。

（4）Command Window：控制在桌面系统中显示或隐藏命令窗口。

（5）Command History：控制在桌面系统中显示或隐藏历史命令窗口。

（6）Current Directory：控制在桌面系统中显示或隐藏当前路径浏览器窗口。

（7）Workspace：控制在桌面系统中显示或隐藏工作空间窗口。

（8）Help：控制在桌面系统中显示或隐藏帮助界面。

（9）Profiler：控制在桌面系统中显示或隐藏调试器界面。

（10）Editor：控制在桌面系统中显示或隐藏 M 文件编辑窗口。

（11）Figures：控制在桌面系统中显示或隐藏图形窗口。

（12）Web Browser：控制在桌面系统中显示或隐藏 Web Browser 窗口。

（13）Variable Editor：控制在桌面系统中显示或隐藏工作空间变量编辑窗口。

（14）File and Directory Comparisons：控制在桌面系统中显示或隐藏文件和目录比较窗口。

（15）Toolbar：控制在桌面系统中显示或隐藏工具栏选项。

（16）Titles：控制在桌面系统中显示或隐藏标题栏选项。

6. Window 主菜单

Window 主菜单能够在所打开的文件或者窗口中，重新设置它们的位置和大小，还可以实现它们之间的快速切换，其下拉菜单项有：

（1）Close All Documents：关闭所有文档，包括 M-file、Figure、Model 和 GUI 窗口。

（2）0 Command Window：选定命令窗口为当前活动窗口。

（3）1 Command History：选定命令历史窗口为当前活动窗口。

（4）2 Current Directory：选定当前路径窗口为当前活动窗口。

（5）3 Workspace：选定工作空间窗口为当前活动窗口。

7. Help 主菜单

Help 主菜单用于提供帮助信息，其下拉菜单项有：

（1）Product Help：显示所有 MATLAB 产品的帮助信息。

（2）Function Browser：启动 MATLAB 帮助。

（3）Using the Desktop：启动 Desktop 的帮助。

（4）Using the Command Window：启动命令窗口的帮助。

（5）Web Resources：显示 Internet 上一些相关的资源网址。

（6）Get Product Trials：申请试用版的 MATLAB 软件。

（7）Check for Updates：检查软件是否更新。

（8）Licensing：授权文件的一些相关操作。

（9）Demos：调用 MATLAB 所提供的范例程序。

（10）Terms of Use：显示 MATLAB 软件中使用的术语。

（11）Patents：显示 MATLAB 软件的专利信息。

（12）About MATLAB：显示有关 MATLAB 的信息。

1.4.3 MATLAB 命令窗口

MATLAB 的命令窗口（Command Window）如图 1-3 所示，它用于 MATLAB 命令的交互操作。

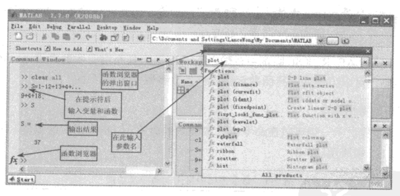


图 1-3 MATLAB 的命令窗口

1. 命令窗口的主要功能和操作

命令窗口具有两大主要功能：

（1）提供用户输入命令的操作平台，用户通过该窗口输入命令和数据。

（2）提供命令执行结果的显示平台，该窗口显示命令执行的结果。

在命令窗口内执行的 MATLAB 主要操作有：

- 运行函数和输入变量。

- 控制输入和输出。
- 执行程序，包括 M 文件和外部程序。
- 保存一段日志。
- 打开或关闭其他应用窗口。
- 各应用窗口的参数选择。

在计算机上安装好 MATLAB 之后，双击 MATLAB 图标，就可以进入命令窗口，此时意味着系统处于准备接受命令的状态，可以在命令窗口中直接输入命令语句。

MATLAB 语句形式为：》变量 = 表达式。

通过等号将表达式的值赋予变量。当输入回车时，该语句被执行。执行之后，窗口自动显示出语句执行的结果。

使用方向键和控制键可以编辑、修改已输入的命令，“↑”键回调上一行命令，“↓”键回调下一行命令。使用“more off”表示不允许分页，“more on”表示允许分页，“more (n)”表示指定每页输出的行数。输入回车键则前进一行，输入空格键显示下一页，输入“q”则结束当前显示。

如果命令语句超过一行或者希望分行输入，则可以使用多行命令分批输入。例如，输入下列式子时，可以通过两行输入。

```
>> S=1-12+13+4+...
9+4+18;
>> S
S =    37
```

说明 三个小黑点是“连行号”，分号“；”的作用是：指令执行结果将不显示在屏幕上，但变量 *S* 将驻留在内存中。

注意，MATLAB R2008b 版本中在输入符“>>”之前新增了函数浏览器（Browse for functions）*fx*，可以方便地进行函数查找以及函数参数的自动帮助。

2．命令窗口的常用命令

MATLAB 提供了一组可以在命令窗口中输入的命令，以执行相应的操作，常用的命令及功能如表 1-1 中所示。

表 1-1 命令窗口中常用的命令及功能

命　令	功　能	命　令	功　能
clc	擦去一页命令窗口，光标回屏幕左上角	pack	整理工作空间内存
clear	清除工作空间中所有的变量	size（变量名）	显示当前工作空间中变量的尺寸
clear all	从工作空间清除所有变量和函数	length（变量名）	显示当前工作空间中变量的长度
clear　变量名	清除指定的变量	disp（变量名）	显示当前工作空间中的变量
clf	清除图形窗口内容	“↑”或“Ctrl+P”	调用上一次的命令
delete <文件名>	从磁盘中删除指定文件	“↓”或“Ctrl+N”	调用下一行的命令

续表

命　　令	功　　能	命　　令	功　　能
help <命令名>	查询所列命令的帮助信息	"←" 或 "Ctrl+B"	退后一格
which <文件名>	查找指定文件的路径	"→" 或 "Ctrl+F"	前移一格
who	显示当前工作空间中所有变量的一个简单列表	Home 或 "Ctrl+A"	光标移到行首
whos	列出变量的大小、数据格式等详细信息	End 或 "Ctrl+E"	光标移到行尾
what	列出当前目录下的 M 文件和 mat 文件	Esc 或 "Ctrl+U"	清除一行
load name	下载'name'文件中的所有变量到工作空间	Del 或 "Ctrl+D"	清除光标后的字符
load name x y	下载'name'文件中的变量 *x*,*y* 到工作空间	Backspace 或 "Ctrl+H"	清除光标前的字符
save name	保存工作空间变量到文件 name.mat 中	Ctrl+K	清除光标至行尾
save name x y	保存工作空间变量 *x*,*y* 到文件 name.mat 中	Ctrl+C	中断程序运行

1.4.4 MATLAB 工作空间

MATLAB 的工作空间如图 1-4 所示。

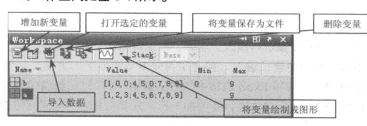


图 1-4　MATLAB 的工作空间

工作空间中的变量以变量名（Name）、数值（Value）和类型（Class）的形式显示出来，双击某个变量，将进入变量编辑器（Variable Editor），可以直接观察变量中具体元素的值，也可以直接修改这些元素。

1．工作空间的工具条

MATLAB 7 的工作空间中还有一个工具条，可快捷地在工作空间中进行许多操作，这些操作在图 1-4 中标注出来了，简单介绍如下。

- （增加新变量）：在工作空间中增加一个新的变量，并可对此变量进行赋值、修改等操作。
- （打开选定的变量）：将工作空间中选定的变量在变量编辑器（Variable Editor）中打开，可对此变量进行修改等操作。
- （导入数据）：将 MATLAB 支持格式的数据导入到工作空间中。
- （将变量保存为文件）：将工作空间中选定的变量以文件的形式保存起来。
- （删除变量）：将工作空间中选定的变量删除。
- （将变量绘制成图形）：将工作空间中选定的变量绘制成图形，支持的绘图函数

有 plot、bar、stem、stairs、area、pie、hist 和 plot3 等。若在工作空间选择某变量后，再单击该按钮，便可实现对该变量的曲线、曲面等图形的绘制。

2. 工作空间的变量编辑器

变量编辑器（Variable Editor）是编辑数组变量的工具，其形式有如 Excel 电子表格，只是它仅能修改及显示，没有计算的功能。在工作空间中选定变量，然后双击，便可进入如图 1-5 所示的变量编辑器窗口。

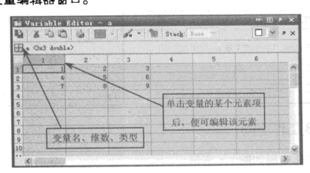


图 1-5　变量编辑器窗口

在编辑器中，可以对变量进行修改、删除、增加等操作，非常方便。需要注意的是：由于大型矩阵不容易由命令窗口输入，因此采用变量编辑器更为方便。变量编辑器可与 Excel 表格的数据相通，只要将 Excel 表格中的数据复制，即可粘贴到编辑器中的某一变量内。原则上，变量的输入以行向为主，要增加一行，只要将其中一元素之位置增加即可，如此即可增加另一行。其余未有数据之空间则以零取代。

3. 工作空间相关的常用命令

MATLAB 还有几个常用的工作空间操作的命令，分别是 who、whos、clear、size、length，其各自功能描述如下。

- who：显示当前工作空间中所有变量的一个简单列表。
- whos：列出变量的大小、数据格式等详细信息。
- clear：清除工作空间中的所有变量。
- clear 变量名：清除指定的变量。
- size(a)：获取向量 ***a*** 的行数与列数。
- length(a)：获取向量 ***a*** 的长度，并在屏幕上显示。如果 ***a*** 是矩阵，则显示的参数为行数中的最大值。

4. 工作空间的数据存取命令

MATLAB 提供了以下保存（save）和载入（load）工作空间的命令。

（1）save 命令

save 命令是将 MATLAB 工作空间中的变量存入磁盘，具体格式介绍如下。

- save：将当前 MATLAB 工作空间中所有变量以二进制格式存入名为 matlab.mat（默认的文件名）的文件中。
- save dfile（文件名）：将当前工作空间中所有变量以二进制格式存入名为 dfile.mat 文件，扩展名自动产生。
- save dfile x：只把变量 x 以二进制格式存入 dfile.mat 文件，扩展名自动产生。
- save dfile.dat x –ascii：将变量 x 以 8 位 ASCII 码形式存入 dfile.mat 文件。
- save dfile.dat x –ascii –double：将变量 x 以 16 位 ASCII 码形式存入 dfile.mat 文件。
- save（fname，'x'，'-ascii'）：fname 是一个预先定义好的包含文件名的字符串，该用法将变量 x 以 ASCII 码形式存入由 fname 定义的文件中，由于在这种用法中，文件名是一个字符变量，因此可以方便地通过编程的方法存储一系列数据文件。

（2）load 命令

load 命令是将磁盘上的数据读入到工作空间，具体格式介绍如下。

- load：把磁盘文件 matlab.mat（默认的文件名）的内容读入内存，由于存储.mat 文件时已包含了变量名的信息，因此调回时已直接将原变量信息带入，不需要重新赋值变量。
- load dfile：把磁盘文件 dfile.mat 的内容读入内存。
- load dfile.dat：把磁盘文件 dfile.mat 的内容读入内存，这是一个 ASCII 码文件，系统自动将文件名（dfile）定义为变量名。
- x=load (fname)：fname 是一个预先定义好的包含文件名的字符串，将由 fname 定义文件名的数据文件读入变量 x 中，使用这种方法可以通过编程方便地调入一系列数据文件。

1.4.5 M 文件编辑/调试器

将 MATLAB 语句按特定的顺序组合在一起就得到了 MATLAB 程序，其文件名的后缀为.m，故也称为 M 文件。MATLAB 7.x 提供了 M 文件的专用编辑/调试器，在编辑器中，会以不同的颜色表示不同的内容：命令、关键字、不完整字符串、完整字符串及其他文本，这样就可以发现输入错误，缩短调试时间。

M 文件编辑/调试器如图 1-6 所示。

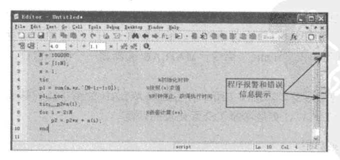


图 1-6 M 文件编辑/调试器

1. M 文件编辑器的特点

MATLAB 编辑器与其他 Windows 编辑程序类似，此处只对下列几点作特别说明。

（1）在编辑 M 文件时，可直接转到指定的行。方法为：从“Go”菜单中选择“Go To”命令。

（2）可直接计算 M 文件中表达式的值，将结果显示在命令窗口中。方法为：选择表达式，然后在“Text”菜单中选择“Evalueate Selection”命令。

（3）可根据 MATLAB 的句法自动缩排，以增加 M 文件的可读性。方法为：先选择文本块，然后按鼠标右键，在“Text”菜单中选择“Smart Indent”命令。

2. 编辑器的工具栏

下面只对此工具栏中特殊的按钮进行叙述，如表 1-2 所示。

表 1-2 工具栏中特殊的按钮

图 标	按钮功能	图 标	按钮功能
	保持文件并以 HTML 格式发布		左右显示 M 文件
	相当于“Edit”菜单中的“Find Next”命令		上下显示 M 文件
	后退一步		浮动 M 文件
	前进一步		最大化 M 文件
	显示函数		计算数组
	设置/取消指定行的断点		计算前面所有数组
	清除所有 M 文件中的断点	- 4.0	指针减小并计算数组
	逐步执行程序	4.0 +	指针增大并计算数组
	进入子函数中逐步执行程序	÷ 1.1	指针被除并计算数组
	跳出子函数	1.1 ×	指针被乘并计算数组
	保存后继续执行所调试的程序		插入数组
	退出调试状态		显示数组标题
	打开函数浏览器		显示数组模式信息
	全部显示 M 文件		

1.4.6 图形窗口

MATLAB 图形窗口（Figure）主要用于显示用户所绘制的图形。通常，只要执行了任意一种绘图命令，图形窗口就会自动产生。绘图都在这一个图形窗口中进行。如果再建一个图形窗口，则可输入“figure”命令，MATLAB 会新建一个图形窗口，并自动给它排出序号。

MATLAB 的图形窗口如图 1-7 所示。它是 MATLAB 绘图功能的基础，使用起来极其方便。其菜单和工具栏，更是增添了交互处理的功能。

1. 图形窗口的菜单栏

图形窗口的“Desktop”（桌面）菜单、“Window”（窗口）菜单和“Help”（帮助）菜

单，与其他系统的大致一样，也比较简单，可以对照学习，在此不再叙述。下面只对差别较大的菜单项进行介绍。

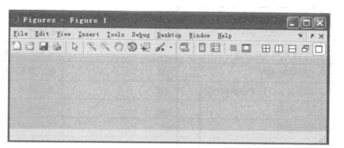


图 1-7 图形窗口

（1）File 菜单

其主要功能命令与桌面平台的 File 菜单相近，只是增加了图形输出 Generate M-file 命令、Export Setup、Print Preview 和 Print 命令。

- Generate M-file 命令：生成当前图形的 M 文件。
- Export Setup 命令：打开 Export Setup（图形输出设置）对话框。
- Print Preview 命令：打开打印预览对话框。

（2）View 菜单

其中的 Figure Toolbar 命令用于控制是否显示图形窗口中的工具栏，而 Camera Toolbar 命令用于控制是否显示图形窗口中的照相操作工具栏。

（3）Insert 菜单

通过该菜单，可以在图形窗口中添加不同的对象，主要有：X Label、Y Label、Z Label、Title、Legend（图例）、Colorbar（颜色条）、Line、Arrow、Text Arrow、Double Arrow、TextBox、Rectangle、Ellipse、Axes 和 Light（光源）等。

（4）Tools 菜单。

此菜单包括简单的图形操作和照相操作。在此只介绍图形操作。

- Basic Fitting 命令：打开图形基本数据拟合对话框。在该对话框中，用户可以根据需要选择拟合的数据源（Select data）、拟合方式（Check to display fits on figure）、拟合函数的显示（Show equations）、数值的有效位数（Significant digits），以及是否显示残差（Plot residuals）和是否显示最大残差模（Show norm of residuals）等。
- Data Statistics 命令：打开图形数据统计分析对话框。在对话框中可以选择数据的最小值（min）、最大值（max）、平均值（mean）、中值（median），以及均方差（std）等。

2. 图形窗口的工具栏

下面只对此工具栏中特殊的按钮进行叙述，如表 1-3 所示。

表 1-3　工具栏各按钮的图标及功能

图　　标	按钮功能	图　　标	按钮功能
	新建一个图形文件		对图形进行三维手动旋转
	打开一个图形文件		数据指针
	以.fig 的格式保存图形文件		将所选的数据点刷成所选的颜色
	打印图形文件		插入颜色工具栏
	使图形窗口处于被编辑状态		插入图例
	放大图形		隐藏绘图工具
	缩小图形		显示绘图工具
	拖动图形		

1.4.7　MATLAB 文件管理

MATLAB 提供了一组文件管理命令，包括列文件名、显示或删除文件、显示或改变当前目录等，相关的命令及其功能如表 1-4 所示。

表 1-4　MATLAB 常用文件管理命令

命　　令	功　　能	命　　令	功　　能
what	显示当前目录与 MATLAB 相关的文件及路径	type filename	在命令窗口中显示文件 filename
dir	显示当前目录下所有的文件	delete filename	删除文件 filename
which	显示某个文件的路径	cd ..	返回上一级目录
cd path	由当前目录进入 path 目录	cd	显示当前目录

1.4.8　MATLAB 帮助

MATLAB 为用户提供了非常丰富的帮助信息，如软件产品帮助（Product Help）、函数帮助（函数浏览器）、网络资源帮助等，极大地完善了该应用软件的功能。

MATLAB 在命令窗口提供了可以获得帮助的命令，用户可以很容易地获得联机帮助信息，几个常用的帮助命令介绍如下。

（1）helpwin：帮助窗口。

（2）helpdesk：帮助桌面，浏览器模式。

（3）lookfor：返回包含指定关键词的项。

（4）demo：打开示例窗口。

MATLAB 还提供了丰富的 help 命令，如表 1-5 所示，在命令窗口中输入相关命令就可以获得相关的帮助。

表 1-5　MATLAB 常用帮助命令

命　　令	功　　能	命　　令	功　　能
help matfun	矩阵函数，数值线性代数	help datafun	数据分析和傅里叶变换函数
help general	通用命令	help ops	操作符和特殊字符

续表

命　令	功　能	命　令	功　能
help graphics	通用图形函数	help polyfun	多项式和内插函数
help elfun	基本的数学函数	help lang	语言结构和调试
help elmat	基本矩阵和矩阵操作	help strfun	字符串函数
help control	控制系统工具箱函数		

1.5 MATLAB 的工具箱

MATLAB 的一个重要特色，就是它具有一套程序扩展系统和一组称之为工具箱（Toolbox）的特殊应用子程序。工具箱是 MATLAB 的关键部分，它是 MATLAB 强大功能得以实现的载体和手段，它是对 MATLAB 基本功能的重要扩充。

MATLAB 的工具箱每年都会有一些变化，要么是出现新的工具箱或实用工具，要么是原有工具箱的性能得到改进。因此，在一般情况下，工具箱的列表不是固定不变的，有关 MATLAB 工具箱的最新信息可以在 http://www.mathworks.com/products 中看到。

MATLAB 有 30 多个工具箱，大致可分为两类：功能型工具箱和领域型工具箱。

（1）功能型工具箱主要用来扩充 MATLAB 的符号计算功能、图形建模仿真功能、文字处理功能，以及与硬件实时交互功能，能用于多种学科。

（2）领域型工具箱专业性很强，是针对某个专业的常用算法做成的函数包，如控制系统工具箱（Control System Toolbox）、信号处理工具箱（Signal Processing Toolbox）、金融工具箱（Financial Toolbox）等。

运行 MATLAB 后，选择“Start”→“Toolboxes”，便会看到按字母顺序列出的 MATLAB 工具箱。

下面简要列举常用计算相关的工具箱所包含的主要内容。

1．最优工具箱（Optimization Toolbox）

（1）线性规划和二次规划
（2）求函数的最大值和最小值
（3）多目标优化
（4）约束条件下的优化
（5）非线性方程求解

2．符号数学工具箱（Symbolic Math Toolbox）

（1）符号表达式和符号矩阵的创建
（2）符号微积分、线性代数、方程求解
（3）因式分解、展开和简化
（4）符号函数的二维图形
（5）图形化函数计算器



3．样条工具箱（Spline Toolbox）

（1）分段多项式和 B 样条

（2）样条的构造

（3）曲线拟合及平滑

（4）函数微积分

1.6 小结

本章首先概要讲述了 MATLAB 的产生和发展历程、其优势及特点，然后一一讲述了 MATLAB 的系统结构、工具箱和桌面操作环境，本章是全书内容的基础，需要扎实掌握。



第 2 章　MATLAB 计算基础

MATLAB 也是一门计算语言，它的运算指令和语法都是基于一系列基本的矩阵运算以及它们的扩展运算的，它还支持复数这一数值元素，这也是 MATLAB 区别于其他高级语言的最大特点之一，它给许多领域的计算带来了极大方便。

2.1 MATLAB 数值类型

MATLAB 包括 4 种基本数据类型，即双精度数组、字符串数组、元胞数组、构架数组。而且数据之间可以相互转化，这为其计算功能开拓了广阔的空间。

1. 变量与常量

变量是数值计算的基本单元。与 C 语言等其他高级语言不同，MATLAB 语言中的变量无须事先定义，一个变量以其名称在语句命令中第一次合法出现而定义，运算表达式变量中不允许有未定义的变量，也不需要预先定义变量的类型，MATLAB 会自动生成变量，并根据变量的操作确定其类型。

（1）MATLAB 变量命名规则

MATLAB 中的变量命名规则如下：

- 变量名区分大小写，因此 *A* 与 *a* 表示的是不同的变量。
- 变量名以英文字母开始，第一个字母后可以使用字母、数字、下画线，但不能使用空格和标点符号。
- 变量名长度不得超过 31 位，超过的部分将被忽略。
- 某些常量也可以作为变量使用，如 i 在 MATLAB 中表示虚数单位，但也可以作为变量使用。

常量是指那些在 MATLAB 中已预先定义其数值的变量，默认的常量如表 2-1 所示。

表 2-1　MATLAB 默认常量

名　称	说　明	名　称	说　明
Pi	圆周率	eps	浮点数的相对误差
INF（或 inf）	无穷大	i（或 j）	虚数单位，定义为 $\sqrt{-1}$
NaN（或 nan）	代表不定值（即 0/0）	nargin	函数实际输入参数的个数
Realman	最大的正实数	nargout	函数实际输出参数的个数
Realmix	最小的正实数	ANS（或 ans）	默认变量名，以应答最近一次操作运算结果

（2）MATLAB 变量的显示

任何 MATLAB 语句的执行结果都可以在屏幕上显示，同时赋值给指定的变量，没有指定变量时，赋值给一个特殊的变量 ans，数据的显示格式由 format 命令控制。format 只影响结果的显示，不影响其计算与存储。MATLAB 总是以双字长浮点数（双精度）来执行所有的运算。如果结果为整数，则显示没有小数；如果结果不是整数，则输出形式有表 2-2 所示的几种形式。

表 2-2　MATLAB 的数据显示格式

格　式	含　义	格　式	含　义
format(short)	短格式（5 位定点数）	format long e	长格式 e 方式
format long	长格式（15 位定点数）	format bank	2 位十进制格式
format short e	短格式 e 方式	format hex	十六进制格式

（3）MATLAB 变量的存取

工作空间中的变量可以用 save 命令存储到磁盘文件中。输入命令“save<文件名>”，将工作空间中全部变量存到“<文件名>.mat”文件中去，若省略“<文件名>”则存入文件 matlab.mat 中；命令“save <文件名> <变量名集>”将“<变量名集>”指出的变量存入文件“<文件名>.mat”中。

用 load 命令可将变量从磁盘文件读入 MATLAB 的工作空间，其用法为“load <文件名>”，它将“<文件名>”指出的磁盘文件中的数据依次读入名称与“<文件名>”相同的工作空间中的变量中去。若省略“<文件名>”则从 matlab.mat 中读入所有数据。

用 clear 命令可从工作空间中清除现存的变量。

2. 字符串

字符是 MATLAB 中符号运算的基本元素，也是文字等表达方式的基本元素，在 MATLAB 中，字符串作为字符数组用单引号（'）引用到程序中，还可以通过字符串运算组成复杂的字符串。字符串数值和数字数值之间可以进行转换，也可以执行字符串的有关操作。

3. 元胞数组

元胞是元胞数组（Cell Array）的基本组成部分。元胞数组与数字数组相似，以下标来区分，元胞数组由元胞和元胞内容两部分组成。用花括号“{ }”表示元胞数组的内容，用圆括号“()”表示元胞元素。与一般的数字数组不同，元胞可以存放任何类型、任何大小的数组，而且同一个元胞数组中各元胞的内容可以不同。

【例 2-1】　元胞数组创建与显示实例。

解：MATLAB 程序代码如下。

```
A(1, 1)={'An example of cell array'};
A(1, 2)={[1 2;3 4]};
A{2, 1}=tf (1, [1, 8]); A{2, 2}={A(1, 2);'This is an example'};
```

```
celldisp(A)  %显示该元胞数组
```

元胞数组 ***A*** 的第 1 行用元胞数组标志法建立一个字符串和一个矩阵；第 2 行用元胞内容标志法，建立一个传递函数和一个由两个元素组成的元胞组，该元胞组分别是矩阵和字符串，最后，用 celldisp 函数显示该元胞数组 ***A***。

4．构架数组

与元胞数组相似，构架数组（Structure Array）也能存放各类数据，使用指针方式传递数值。构架数组由结构变量名和属性名组成，用指针操作符"."连接结构变量名和属性名。例如，可用 parameter.temperature 表示某一对象的温度参数，用 parameter.humidity 表示某一对象的湿度参数等，因此，该构架数组 parameter 由两个属性组成。

5．对象

面向对象的 MATLAB 语言采用了多种对象，如自动控制中常用的传递函数模型对象（tf object）、状态空间模型对象（ss object）和零极点模型对象（zpk object），一些对象之间可以相互转换，例如可以从传递函数模型对象转化为零极点模型对象，这将在后面具体介绍。

2.2 关系运算和逻辑运算

除了传统的数学运算外，MATLAB 还支持关系运算和逻辑运算。如果你已经有了一些编程经验，对这些运算不会陌生。这些操作符和函数的目的是提供求解真/假命题的答案。关系运算和逻辑运算主要用于控制基于真/假命题的各 MATLAB 命令（通常在 M 文件中）的流程或执行次序。

作为所有关系表达式和逻辑表达式的输入，MATLAB 把任何非 0 数值当做真，把 0 当做假。所有关系表达式和逻辑表达式，对于真，输出为 1；对于假，输出为 0。

MATLAB 为关系运算和逻辑运算提供了关系操作符和逻辑操作符，如表 2-3 和表 2-4 所示。

表 2-3　关系运算符

符　号	功　能	符　号	功　能
<	小于	>=	大于或等于
<=	小于或等于	==	等于
>	大于	~=	不等于

表 2-4　逻辑运算符

符　号	功　能	符　号	功　能
&	逻辑与	~	逻辑非
|	逻辑或		

此外，MATLAB 还提供了若干关系运算函数和逻辑运算函数，分别如表 2-5 和表 2-6

所示。

表 2-5　关系运算函数

函 数 名	功　　能	函 数 名	功　　能
all	所有向量为非零元素时为真	xor	逻辑异或运算
any	任一向量为非零元素时为真		

表 2-6　逻辑运算函数

函 数 名	功　　能	函 数 名	功　　能
bitand	位方式的逻辑与运算	bitcmp	位比较运算
bitor	位方式的逻辑或运算	bitmax	最大无符号浮点整数
bitxor	位方式的逻辑异或运算	bitshift	将二进制移位运算

2.3 矩阵及其运算

MATLAB 软件的最大特色是强大的矩阵计算功能，在 MATLAB 软件中，所有的计算都是以矩阵为单元进行的，可见矩阵是 MATLAB 的核心。下面以表格的形式列出 MATLAB 提供的每类矩阵运算的函数，并各举一个实例进行说明，同类函数的用法基本类似，详细的用法及函数内容说明可参考联机帮助。

2.3.1 矩阵的创建

由 m 行 n 列构成的数组 $\boldsymbol{A}$ 称为 $m\times n$ 阶矩阵，它总共由 $m\times n$ 个元素组成，矩阵元素记为 a_{ij}，其中 i 表示行，j 表示列。

当 $m=n$ 时，矩阵 $\boldsymbol{A}$ 称为方阵。当 $i\neq j$ 时，所有的 $a_{ij}=0$，且 $m=n$，得到的矩阵称为对角阵。

当对角阵的对角线上的元素全为 1 时，称为单位阵，记为 $\boldsymbol{I}$。

对于 $(m\times n)$ 阶矩阵 $\boldsymbol{W}$，当 $w_{ij}=a_{ji}$ 时，称 $\boldsymbol{W}$ 是 $\boldsymbol{A}$ 的转置矩阵，记为 $\boldsymbol{W}=\boldsymbol{A}'$。

对于 $\boldsymbol{A}$ 为 $(m\times 1)$ 的形式时，称 $\boldsymbol{A}$ 是 m 个元素的列向量，对于 $\boldsymbol{A}$ 为 $(1\times n)$ 的形式时，称 $\boldsymbol{A}$ 是 n 个元素的行向量。

矩阵的表现形式和数组相似，它以左方括号“[”开始，以右方括号“]”结束，每一行元素结束用行结束符号（分号“;”）或回车符分割，每个元素之间用元素分割符号（空格或“,”）分隔。建立矩阵的方法有直接输入矩阵元素、在现有矩阵中添加或删除元素、读取数据文件、采用现有矩阵组合、矩阵转向、矩阵移位，以及直接建立特殊矩阵等。

【例 2-2】　矩阵创建实例。

解： MATLAB 程序代码如下。

```
>> A=[1 2 3;4 5 6]
```

运行结果是创建了一个 2×3 的矩阵 $\boldsymbol{A}$，$\boldsymbol{A}$ 的第 1 行由 1、2、3 这 3 个元素组成，第 2

行由 4、5、6 这 3 个元素组成，输出结果如下：

```
A = 1    2    3
    4    5    6
```

接着输入：

```
>> B=[A; 11, 12, 13]  %添加一行元素[11, 12, 13]
```

运行结果是创建了一个 3×3 的矩阵 ***B***，***B*** 矩阵是在 ***A*** 矩阵的基础上添加一行元素 11、12、13，组成一个 3×3 矩阵，输出结果如下：

```
b = 1    2    3
    4    5    6
    11   12   13
```

MATLAB 中对矩阵元素的访问如下所示：

- 单个元素的访问：***B***(3, 2)→12，访问了第 3 行和第 2 列交叉的元素。
- 整列元素的访问：***B***(:, 3)→[3, 6, 13]'，访问了第 3 列中的所有元素。
- 整行元素的访问：***B***(1, :)→[1, 4, 11]，访问了第 1 行中的所有元素。
- 整块元素的访问：***B***(2:3, 2:3)→[5, 6; 12, 13]，访问了一个 2×2 的子块矩阵。

MATLAB 提供了很多个特殊矩阵的生成函数，表 2-7 列出了一些常用的生成函数，关于其他的特殊矩阵生成函数及使用格式，请参见联机帮助。

表 2-7 MATLAB 常用特殊矩阵生成函数

函 数	功能说明	函 数	功能说明
zeros()	生成元素全为 0 的矩阵	tril()	生成下三角矩阵
ones()	生成元素全为 1 的矩阵	eye()	生成单位矩阵
rand()	生成均匀分布随机矩阵	company()	生成伴随矩阵
randn()	生成正态分布随机矩阵	hilb()	生成 Hilbert 矩阵
magic()	生成魔方矩阵	vander()	生成 Vander 矩阵
diag()	生成对角矩阵	hankel()	生成 Hankel 矩阵
triu()	生成上三角矩阵	hadamard()	生成 Hadamard 矩阵

【例 2-3】 特殊矩阵生成函数使用实例。

解： MATLAB 程序代码如下。

```
>> A=[1, 2, 3; 4, 5, 6; 7, 8, 9];  B=tril(A)  %生成下三角矩阵
```

运行结果是生成了 ***B*** 矩阵，它是调用下三角矩阵生成函数 tril()生成的矩阵 ***A*** 的下三角矩阵，输出结果如下：

```
B = 1    0    0
    4    5    0
    7    8    9
```

2.3.2 矩阵的运算

矩阵与矩阵之间可以进行如表 2-8 所示的基本运算。

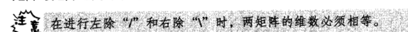

在进行左除“/”和右除“\”时，两矩阵的维数必须相等。

表 2-8　矩阵基本运算

操作符号	功能说明	操作符号	功能说明
+	矩阵加法	/	矩阵的左除
−	矩阵减法	'	矩阵转置
*	矩阵乘法	logm( )	矩阵对数运算
^	矩阵的幂	expm( )	矩阵指数运算
\	矩阵的右除	inv( )	矩阵求逆

【例 2-4】　矩阵基本运算实例。

解：MATLAB 程序代码如下。

```
>> A=[1, 2; 3, 4]; B=[3, 5; 2, 9]; div1=A/B; %矩阵的左除
>>div2=B\A  %矩阵的右除
```

两个矩阵 ***A*** 与 ***B*** 进行了左除和右除运算，输出结果如下：

```
div1 =                          div2=
      0.2941    0.0588               -0.3529   -0.1176
      1.1176   -0.1765                0.4118    0.4706
```

MATLAB 提供了多种关于矩阵的函数，表 2-9 列出了一些常用的矩阵函数运算。

表 2-9　常用矩阵函数运算

函 数 名	功能说明	函 数 名	功能说明
rot90()	矩阵逆时针旋转 90°	eig()	计算矩阵的特征值和特征向量
flipud()	矩阵上下翻转	rank()	计算矩阵的秩
fliplr()	矩阵左右翻转	trace()	计算矩阵的迹
flipdim()	矩阵的某维元素翻转	norm()	计算矩阵的范数
shiftdim()	矩阵的元素移位	poly()	计算矩阵的特征方程的根

【例 2-5】　矩阵函数运算实例。

解：MATLAB 程序代码如下。

```
>> A=[1, 3, 5; 2, 4, 6; 7, 9, 13]; [B, C]=eig(A)  %求取矩阵的特征值和特征向量
```

通过函数 eig( )计算矩阵 ***A*** 的特征向量 ***b*** 和特征值 ***c***，输出结果如下：

```
B=-0.3008   -0.7225    0.2284
  -0.3813   -0.3736   -0.8517
  -0.8742    0.5817    0.4717
C = 19.3341       0         0
```

```
0       -1.4744       0
0             0  0.1403
```

矩阵分解常用于方程求根，表 2-10 列出了一些常用的矩阵分解运算。

表 2-10 常用矩阵分解运算函数

函 数 名	功能说明	函 数 名	功能说明
eig()	矩阵的特征值分解	svd()	矩阵的奇异值分解
qr()	矩阵的 QR 分解	chol()	矩阵的 Cholesky 分解
schur()	矩阵的 Schur 分解	lu()	矩阵的 LU 分解

【例 2-6】 矩阵分解运算实例。

解： MATLAB 程序代码如下。

```
>> A=[6, 2, 1; 2, 3, 1; 1, 1, 1];  [L, U, P]=lu(a)  %对矩阵进行 LU 分解
```

通过函数 lu()对矩阵 $\boldsymbol{A}$ 进行 LU 分解，得到上三角阵 $\boldsymbol{U}$、下三角阵 $\boldsymbol{L}$、置换矩阵 $\boldsymbol{P}$，输出结果如下：

```
L =1.0000        0        0          U =  6.0000   2.0000   1.0000
   0.3333   1.0000        0                    0   2.3333   0.6667
   0.1667   0.2857   1.0000                    0        0   0.6429
P = 1    0    0
    0    1    0
    0    0    1
```

2.4 复数及其运算

复数以及在其基础上发展起来的复变函数这一重要的数学分支，解决了许多实变函数无法解决的问题，有着广泛的工程应用。

复变函数是控制工程的数学基础，常用来描述控制系统模型的传递函数，就是属于复变函数。MATLAB 支持在运算和函数中使用复数或复数矩阵，还支持复变函数运算。

2.4.1 复数表示

MATLAB 是以 i 或 j 字元来代表虚部复数运算的。

一个复数可表示为：$x=a+bi$，其中 a 称为实部，b 称为虚部。

也可写成复指数形式：$x=re^{i\theta}$。其中 r 称为复数的模，又记为 $|x|$；θ 称为复数的幅角，又记为 $Arg(x)$，且满足如下关系：

$$r=\sqrt{a^2+b^2}\text{，}\tan\theta=\frac{b}{a}$$

具体采用哪种表示方法取决于实际问题。一般而言，第一种问题适合处理复数的代数运算，第二种方法适合处理复数旋转等涉及幅角改变的问题。

一个复数也可以看做是关于实部和虚部的符号函数。因此，既可以直接构造复数，也可以用符号函数构造复数。下面分别介绍这两种构造方法。

（1）用直接法构造两种形式的复数

直接法就是利用符号 i 或 j 来表示复数单位，将复数看做完整的一个表达式。其具体形式也有两种，可以用实部和虚部形式表示，也可以用复指数形式表示。

（2）用符号函数法构造两种形式的复数

所谓符号函数法就是将复数看成是函数形式，将其实部和虚部看做自变量，用 syms 来构造，用 subs 对符号函数中自变量进行赋值。

【例 2-7】 复数构造实例。试分别使用上述两种方法，在 MATLAB 中构造复数 $x=-1+\mathrm{i}$。

解：在 MATLAB 命令窗口中输入：

```
x1=-1+i  %直接法构造,实部虚部形式
x2=sqrt(2)*exp(i*(3*pi/4))  %直接法构造,复指数形式
%符号函数法构造,实部虚部形式
syms a b real;        %声明 a, b 为实数型
x3=a+b*i %实部、虚部形式复数的符号表达
subs(x3,{a,b},{-1,1}) %代入具体值
%符号函数法构造,复指数形式
syms r ct real; %声明 r, ct 为实数型
x4=r*exp(ct*i); %复指数形式复数的符号表达
subs(x4,{r,ct},{sqrt(2),3*pi/4})%代入具体值
```

输出结果为：

```
x1 =  -1.0000 + 1.0000i
x2 =  -1.0000 + 1.0000i
x3 =  -1.0000 + 1.0000i
x4 =  -1.0000 + 1.0000i
```

MATLAB 中矩阵运算贯穿始终，复数矩阵运算也是其中的一个重要方面。由于复数矩阵的每个元素都是复数，所以对应的创建复数矩阵的方法也有两种，即直接创建和利用符号函数创建。

复数矩阵的直接创建有两种方法，一种是由复数元素，构造复数矩阵，另一种是利用两个矩阵分别做实部和虚部构造新的复数矩阵。下面举例说明复数矩阵的直接创建，另一种方法类比很容易实现，在此不作赘述。

（1）由复数元素构造复数矩阵

复数矩阵的每个元素都是复数，将复数作为矩阵元素，并按照矩阵格式进行填充就得到了复数矩阵。

（2）由实矩阵创造复数矩阵

由两个分别作为实部和虚部的同维矩阵构造复数矩阵。

【例 2-8】 复数矩阵构造实例。构造复数矩阵$\begin{bmatrix}1+\mathrm{i} & 1+2\mathrm{i} & 1+3\mathrm{i}\\ 1-\mathrm{i} & 1-2\mathrm{i} & 1-3\mathrm{i}\end{bmatrix}$。

解： 可直接输入各元素来构造矩阵，此处通过其复指数形式构造。

在 MATLAB 命令窗口中输入：

```
%直接输入各元素来构造
A1=[sqrt(2)*exp((pi/4)*i) 1+2i 1+3i;sqrt(2)*exp((-pi/4)*i) 1-2i 1-3i] %
%由实矩阵构造
A2re=[1 1 1;1 1 1]; A2im=[1 2 3;-1 -2 -3]; A2=A2re+A2im*i
```

输出结果 A1 和 A2 均为：

```
1.0000 + 1.0000i   1.0000 + 2.0000i   1.0000 + 3.0000i
1.0000 - 1.0000i   1.0000 - 2.0000i   1.0000 - 3.0000i
```

2.4.2 复数绘图

对于复数函数的绘图主要有两种形式。一种是直角坐标图，即分别以复数的实部和虚部为坐标作出复数的表示图；另一种为极坐标图，即分别以复数的模和幅角为坐标作图。

MATLAB 提供了绘制极坐标图的函数 polar，它还可绘制出极坐标栅格线，其常用的调用格式为：

```
polar(theta,rho)
```

其中，theta 为极坐标极角，rho 为极坐标矢径。

【例 2-9】 复数函数绘图实例。作出函数 $y = t + \mathrm{i}t\sin(t)$ 在两种坐标下的表示图。

解： 直角坐标图——以实部为横坐标，以虚部为纵坐标；极坐标图——以模为极半径，以幅角为极角，在 MATLAB 命令窗口中输入以下命令，最终显示结果如图 2-1 所示。

```
t=0:0.01:2*pi; y=t+i*t.*sin(t); %直角坐标表示
r=abs(y);bdelta=angle(y) %极坐标表示
subplot(2,1,1)
plot(y)  %绘制直角坐标图
title('直角坐标图') ; subplot(2,1,2)
polar(delta,r) %绘制极坐标图
title('极坐标图')
```

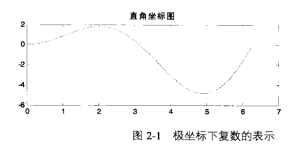


图 2-1 极坐标下复数的表示

2.4.3 复数操作函数

对于一个形如 $x=a+bi$ 的复数，我们通常希望能够了解它自身的结构性质，包括实部、虚部、模和幅角等。MATLAB 提供了方便的操作函数，如表 2-11 所示。

表 2-11 常用矩阵分解运算函数

函数名	功能	函数名	功能
real(*A*)	求复数或复数矩阵 *A* 的实部	abs(*A*)	求复数或复数矩阵 *A* 的模
imag(*A*)	求复数或复数矩阵 *A* 的虚部	angle(*A*)	求复数或复数矩阵 *A* 的相角，单位为弧度
conj(*A*)	求复数或复数矩阵 *A* 的共轭		

在 MATLAB 中，复数的基本运算和实数相同，都是使用相同的函数。比如复数或复数矩阵 **x** 除以 **y**，运算命令也是 ***x*/*y***，与实数运算一样。因此，复数基本运算的内容此处不再赘述。

2.5 符号运算

MATLAB 提供了符号数学工具箱（Symbolic Math Toolbox），大大增强了 MATLAB 的功能。符号数学工具箱的特点为：

（1）用途广泛，而不仅针对一些特殊专业或专业分支。

（2）使用字符串来进行符号分析，而不是基于数组的数值分析。

2.5.1 符号运算概述

符号数学工具箱是操作和解决符号表达式的符号数学工具（函数）集合，有复合、简化、微分、积分，以及求解代数方程和微分方程的工具，另外还有一些用于线性代数的工具，求解逆、行列式、正则形式的精确结果，找出符号矩阵的特征值而没有由数值计算引入的误差。工具箱还支持可变精度运算，即支持符号计算并能以指定的精度返回结果。

符号运算与数值运算的主要区别如下：

（1）数值运算中必须先对变量赋值，然后才能参与运算。

（2）符号运算无须事先对独立变量赋值，运算结果以标准的符号形式表达。

符号运算的运算对象可以是没赋值的符号变量，可以获得任意精度的解。

1. 符号表达式

符号表达式是代表数字、函数、算子和变量的 MATLAB 字符串，或字符串数组。不要求变量有预先确定的值，符号方程式是含有等号的符号表达式。符号算术是使用已知的规则和给定符号恒等式求解这些符号方程的实践，它与代数和微积分中的求解方法完全一样。符号矩阵的元素是符号表达式。

MATLAB 在内部把符号表达式表示成字符串，与数字变量或运算相区别；否则，这些符号表达式几乎与基本的 MATLAB 命令毫无区别。

2．创建符号对象

MATLAB 提供了两个建立符号对象的函数：sym 和 syms，这两个函数的用法介绍如下。

（1）sym 函数

sym 函数用来建立单个符号量，一般调用格式为：

```
符号量名=sym('符号字符串')
```

该函数可以建立一个符号量，符号字符串可以是常量、变量、函数或表达式。应用 sym 函数还可以定义符号常量。

（2）syms 函数

函数 sym 一次只能定义一个符号变量，使用起来不方便。MATLAB 提供了另一个函数 syms，一次可以定义多个符号变量。

syms 函数的一般调用格式为：

```
syms 符号变量名 1 符号变量名 2 … 符号变量名 n
```

用这种格式定义符号变量时不要在变量名上加字符串分界符（'），变量间用空格而不要用逗号分隔。

含有符号对象的表达式称为符号表达式，建立符号表达式有以下 3 种方法：

- 利用单引号来生成符号表达式。
- 用 sym 函数建立符号表达式。
- 使用已经定义的符号变量组成符号表达式。

【例 2-10】 符号表达式创建实例。

解： 在 MATLAB 窗口，输入下列命令：

```
U=sym('3*x^2+5*y+2*x*y+6')          %定义符号表达式
syms x y;                           %建立符号变量 x、y
V=3*x^2+5*y+2*x*y+6                 %定义符号表达式 V
2*U-V+6                             %求符号表达式的值
```

3．符号矩阵

符号矩阵也是一种符号表达式，所以前面介绍的符号表达式运算都可以在矩阵意义下进行。但应注意这些函数作用于符号矩阵时，是作用于矩阵的每一个元素。

通过 sym 函数创立符号矩阵，矩阵元素可以是任何不带等号的符号表达式，各符号表达式的长度可以不同，矩阵元素之间可用空格或逗号分隔。

例如，在命令窗口中输入“A = sym('[a , 2*b ; 3*a , 0]')”，就完成了一个符号矩阵 $\boldsymbol{A}=\begin{bmatrix} a & 2b \\ 3a & 0 \end{bmatrix}$ 的创建，输出的结果为：

```
A = [ a, 2*b]
    [3*a, 0]
```

需要注意的是：符号矩阵的每一行的两端都有方括号，这是与 MATLAB 数值矩阵的一个重要区别。

在 MATLAB 中，数值矩阵不能直接参与符号运算，必须先转化为符号矩阵。

（1）将数值矩阵转化为符号矩阵

函数调用格式：sym(数值矩阵)

（2）将符号矩阵转化为数值矩阵

函数调用格式：numeric(*A*)

由于符号矩阵是一个矩阵，所以符号矩阵还能进行有关矩阵的运算。MATLAB 还有一些专用于符号矩阵的函数，这些函数作用于单个的数据无意义。例如：

transpose(***S***)：返回 ***S*** 矩阵的转置矩阵。

determ(***S***)：返回 ***S*** 矩阵的行列式值。

其实，许多应用于数值矩阵的函数，如 diag、triu、tril、inv、det、rank、eig 等，也可直接应用于符号矩阵。

4．符号表达式的四则运算

符号表达式的四则运算比较简单，常用的函数介绍如下。

- factor(***S***)：对 ***S*** 分解因式，***S*** 是符号表达式或符号矩阵。
- expand(***S***)：对 ***S*** 进行展开，***S*** 是符号表达式或符号矩阵。
- collect(***S***)：对 ***S*** 合并同类项，***S*** 是符号表达式或符号矩阵。
- collect(***S***,*v*)：对 ***S*** 按变量 *v* 合并同类项，***S*** 是符号表达式或符号矩阵。
- simplify(***S***)：应用函数规则对 ***S*** 进行化简。
- simple(***S***)：调用 MATLAB 的其他函数对表达式进行综合化简，并显示化简过程。

2.5.2 常用的符号运算

符号变量和数字变量之间可转换，也可以用数字代替符号得到数值。符号运算种类非常多，常用的符号运算有代数运算、积分和微分运算、极限运算、级数求和、方程求解等。

出于篇幅的考虑，下面仅对最优化计算中常用的符号运算进行介绍，其他的符号运算的使用方法大同小异，读者可通过 MATLAB 的帮助文档或其他关于符号函数工具箱的书籍进行学习，此处不再赘述。

常用的符号运算有求极值、级数求和、微积分、解微分方程等，下面分别进行介绍。

1．limit

limit 是求极限的符号函数，其常用的格式是：

`limit(F, x, a, 'right')` 或 `limit(F, x, a, 'left')`

表示当自变量 *x* 从右侧或左侧逼近 *a* 时，函数 *F* 的极值。

2. diff

diff 是求微分最常用的符号函数，其输入参数既可以是函数表达式，也可以是符号矩阵。

常用的格式是：diff(*f*, *x*, *n*)

表示 *f* 关于 *x* 求 *n* 阶导数。

3. int

int 是求积分最常用的符号函数，其输入参数可以是函数表达式。

常用的格式是：int(*f*, *r*, *x*0, *x*1)

其中，*f* 为所要积分的表达式，*r* 为积分变量，若为定积分，则 *x*0 与 *x*1 为积分上下限。

4. symsum

symsum 是级数求和的符号函数，其常用的格式是：

```
S=symsum(fk, k, k0, kn)
```

其中 fk 为级数的通项，*k* 为级数自变量，*k*0 和 *k*n 为级数求和的起始项和终止项，且可设为 inf。

5. dsolve

dsolve 求解常微分方程的符号函数，其常用的格式是：

```
dsolve('eqn1', 'condition', 'var')
```

该函数求解微分方程 eqn1 在初值条件 condition 下的特解。参数 var 描述方程中的自变量符号，省略时按默认原则处理，若没有给出初值条件 condition，则求方程的通解。

dsolve 在求微分方程组时的调用格式为：

```
dsolve('eqn1', 'eqn2', …, 'eqnN', 'condition1', …, 'conditionN', 'var1', …,
'varN')
```

函数求解微分方程组 eqn1, …, eqnN 在初值条件 conditoion1, …, conditionN 下的解，若不给出初值条件，则求方程组的通解，var1, …, varN 给出求解变量。

【例 2-11】 极限和极值的符号运算实例。求极限 $f1=\lim\limits_{x\to 1^+}\left[\dfrac{1}{x\ln^2 x}-\dfrac{1}{(x-1)^2}\right]$，求和 $f2=\sum\limits_{n=0}^{50}\left[an^3+(a-1)n^2\right]$。

解：输入如下 MATLAB 程序代码。

```
syms x  a  n  %定义符号变量 a 和 x
y=1/(x*(log(x)^2))-1/(x-1)^2;
f1=limit(y,x,1,'right')  %求极限
f=a*n^3+(a-1)*n^2
f2=symsum(f,n,0,50)  %级数求和
```

运行程序，输出结果如下所示：

```
f1 =1/12
f2 =1668550*a - 42925
```

可知 $\lim_{x\to 1^+}\left[\frac{1}{x\ln^2 x}-\frac{1}{(x-1)^2}\right]=\frac{1}{12}$，$\sum_{n=0}^{50}\left[an^3+(a-1)n^2\right]=-42925+1668550*a$。

【例 2-12】 微积分的符号运算实例。(1) 已知表达式 $f=\sin(ax)$，分别对其中的 x 和 a 求导；(2) 已知表达式 $f=x\log(1+x)$，求对 x 的积分和 x 在(0,1)上的积分值。

解：输入如下 MATLAB 程序代码。

```
syms a x                    %定义符号变量 a 和 x
f=sin(a*x)                  %创建函数 f
dfx=diff(f, x)              %对 x 求导
dfa=diff(f, a)              %对 a 求导
f1=x*log(1+x)               %创建函数 f1
int1=int(f1,x)              %对 x 积分
int2=int(f1,x,0,1)          %求[0,1]区间上的积分
```

运行程序，输出结果如下所示：

```
dfx =cos(a*x)*a                 %f 对 x 求导的结果
dfa =cos(a*x)*x                 %f 对 a 求导的结果
int1 =x/2 - x^2/4 + (log(x + 1)*(x^2 - 1))/2  %积分表达式
int2 =1/4                       %积分值
```

【例 2-13】 常微分方程符号运算实例。(1) 计算微分方程 $\frac{\mathrm{d}y}{\mathrm{d}x}+3xy=x\mathrm{e}^{-x^2}$ 的通解。(2) 计算微分方程 $xy'+2y-\mathrm{e}^x=0$ 在初始条件 $y|_{x=1}=2\mathrm{e}$ 下的特解。(3) 求 $y''+2y'+\mathrm{e}^x=0$ 的通解。

解：输入如下 MATLAB 程序代码。

```
f1=dsolve('Dy+3*x*y=x*exp(-x^2)','x')
f2=dsolve('x*Dy+2*y-exp(x)=0','y(1)=2*exp(1)','x')
f3=dsolve('D2y+2*Dy+exp(x)=0','x')
```

运行程序，输出结果如下所示：

```
f1=C9/exp((3*x^2)/2) + exp(x^2/2)/exp((3*x^2)/2)
f2 =(2*exp(1))/x^2 + (exp(x)*(x - 1))/x^2
f3 =C7/exp(2*x) - (exp(3*x)/3 + (C6*exp(2*x))/2)/exp(2*x)
```

可知(1)的通解为 $y=\mathrm{e}^{-x^2}+\mathrm{e}^{-\frac{3}{2}x^2}*C_9$，其中 C_9 为常数；(2)的特解为 $y=\frac{x\mathrm{e}^x-\mathrm{e}^x+2\mathrm{e}}{x^2}$；(3) 的通解为 $y=-\frac{1}{3}\mathrm{e}^x-C_7*\mathrm{e}^{-2x}+C_6$，其中 C_7，C_6 为常数。

2.6 小结

MATLAB 语言具有强大的数值计算功能，熟练掌握和运用这些功能是发挥 MATLAB 强大功能的基础。



第 3 章 MATLAB 绘图入门

俗话说“一图胜万语”，在科学研究、工程上有图则一目了然，无图搭配则如隔靴搔痒，很难得窥得全貌，这也是一般工作偏重于图说的原因。

绘图的目的如果是显示数据的走向或变化趋势，则可采用不同的观察角度，使数据的内容更加明晰。就图的特性分类，可以分为块状图、柱状图、点示图、线示图等，而就其空间而言，又可分为二维或三维图，前者取其实用性，后者取其美观性。

MATLAB 提供了强大的图形功能，利用程序与绘图结合，可以将计算结果以图形显现，有助于了解计算过程以及分析计算结果，这一点在科学、工程中都非常重要。

3.1 MATLAB 中绘图的基本步骤

在 MATLAB 中绘制图形，通常采用以下 7 个步骤：

1. 准备数据

准备好绘图需要的横坐标变量和纵坐标变量数据。

2. 设置当前绘图区

在指定的位置创建新的绘图窗口，并自动以此窗口的绘图为当前绘图区。

3. 绘制图形

创建坐标轴，指定叠加绘图模式，绘制函数曲线。

4. 设置图形中曲线和标记点格式

设置图形中的线宽、线型、颜色和标记点的形状、大小、颜色等。

5. 设置坐标轴和网格线属性

将坐标轴的范围设置在指定曲线。

6. 标注图形

对图形进行标注，包括在图形中添加标题、坐标轴标注、文字标注等。

7. 保存和导出图形

按指定文件格式、属性保存或导出图形，以备后续使用。

上述绘制流程中，需要注意以下两点：

（1）上面的 7 个步骤的顺序也不是完全固定，尤其是其中对图形进行修饰标注的第 4、

5、6 步骤，完全可以改变顺序。

（2）MATLAB 中对于图形中的曲线和标记点格式有默认的设置，这在一般情况下是可以满足使用者需要的，因此对于只是想大概查看一下数据分布的用户，只需要进行第 1 步和第 3 步工作就可以了。

3.2 在工作空间直接绘图

在 MATLAB 中的，还有一种较为简单的绘图方法，就是直接利用工作空间的数据绘出想要的图形。这种方法使用起来非常简单，只需要单击鼠标选中你要的绘图的类型就可以了。

这种绘图方法的基本步骤是：在工作空间中，首先用鼠标左键，选中要绘制图形的数据变量，看到变量变成蓝颜色后，然后单击工作空间的 [按钮图标] 按钮，并且选择图形的类型，就可以绘出想要的图形了。

如果绘制的是多变量数据，使用 Shift 键全部选中后，再单击绘图图表的图形类别，就可以了。MATLAB 根据变量列出不同种类的图形类别包括 plot、bar、stem、stairs、area、pie、hist 和其他类型图形。

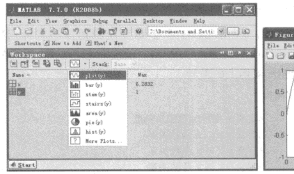
【例 3-1】 工作空间直接作图法使用实例。利用工作空间绘制 $y=\sin x$ 正弦曲线。

解：在命令窗口中输入以下命令：

```
x=-2*pi:pi/100:2*pi;        %定义 x 的范围及刻度
y=sin(x);                   %定义 y 与 x 之间的函数关系
```

运行后，在工作空间中将生成变量 x 和 y：

在工作空间中，可以看到数据名、数据类型、数据最小值和数据最大值，然后鼠标右键单击变量 y，则数据变成蓝颜色，如果此时不选中变量 x，直接单击 [按钮图标] 后，选择“plot(y)”便可绘制图形。操作界面及绘制的图形如 3-1 所示。



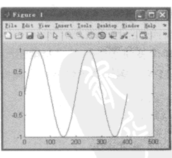


图 3-1 $y=\sin(x)$单变量工作空间图形

如果选中 y 以后，按住 Shift 键，继续选中 x 后，再选择 plot(y)便可绘制图形。操作界面及绘制的图形如 3-2 所示。读者可以比较出两图的差异。

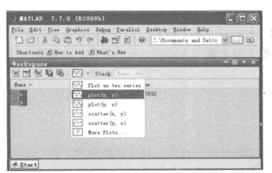
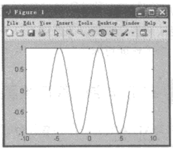



图 3-2　y=sin(x)双变量工作空间波形图

3.3 利用绘图函数绘图

MATLAB 提供了丰富的绘图功能，在命令窗口中输入“help graph2d”可得到所有画二维图形的命令；输入“help graph3d”可得到所有画三维图形的命令。

3.3.1 绘制二维图形

二维图形的基本绘图命令是：

```
plot(x1, y1, option1, x2, y2, option2, …)
```

其中，$x1$ 与 $y1$ 给出的数据分别为 x 轴与 y 轴坐标值，option1 为选项参数，以逐点连折线的方式绘制 1 个二维图形；同时类似地绘制第二个二维图形。

这是 plot 命令的完全格式，在实际应用中可以根据需要进行简化。

比如 plot(x, y)、plot(x, y, option)，选项参数 option 定义了图形曲线的颜色（用颜色英文单词的第一个字母表示，例如 r 表示红色、g 表示绿色、b 表示蓝色）、线型（例如#、*等）及标示符号，它由一对单引号括起来。

【例 3-2】　二维图形绘制实例。利用 plot(x)和多组变量的语法格式分别绘制当 $x \in [0, 2\pi]$ 时，$y1 = \sin(x)$、$y2 = \cos(x)$、$y3 = \sin(x - 0.1\pi)$、$y4 = \cos(x + 0.1\pi)$ 的二维图形，并比较单个变量图形和多组变量图形应用上的差异。

解：在 M 文件编辑器中输入以下命令：

```
x=0:0.4*pi:2*pi;              %定义 x 坐标轴范围及刻度
y1=sin(x);                    %定义 y1 与 x 函数关系
y2=cos(x);                    %定义 y2 与 x 函数关系
y3=sin(x-0.1*pi);             %定义 y3 与 x 函数关系
y4=cos(x+0.1*pi);             %定义 y4 与 x 函数关系
plot(y1)                      %绘制 y1 与 x 函数的图形，如图 3-3
```

运行以上 M 代码程序，得到图 3-3 所示的结果图形。

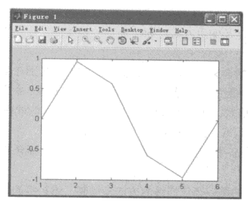


图 3-3 plot(y1)画线结果

如果将程序中 plot(y1)替换成以下语句，即将 3 条曲线绘制在同一图中，将会得到图 3-4 所示结果图形。

```
plot(x,y1,x,y2,x,y3,x,y4)          %图 3-4，注意比较和图 3-3 的不同
```

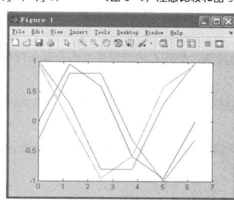


图 3-4 对多组数据应用 plot 画图的结果

3.3.2 绘制三维图形

1. 三维曲线

MATLAB 也提供了一个绘制三维折线或曲线的基本命令 plot3，其常用的格式是：

```
plot3(x1, y1, z1, option1, x2, y2, z2, option2, ….)
```

该命令中：

- 以 x1、y1、z1 所给出的数据分别为 x、y、z 坐标值。
- option1 为选项参数，以逐点连折线的方式绘制 1 个三维折线图形。
- 以 x2、y2、z2 所给出的数据分别为 x、y、z 坐标值。
- option2 为选项参数，以逐点折线的方式绘制另一个三维折线图形。

需要注意的是：

（1）plot3 命令的功能及使用方法与 plot 命令的相类似，它们的区别在于前者绘制出的是三维图形。

（2）plot3 命令参数的含义与 plot 命令的类似，它们的区别在于前者多了一个 z 方向上的参数。而且 plot3 各个参数的取值情况及其操作效果也与 plot 命令相同。上面给出的 plot3 命令格式是一种完整的格式，在实际操作中，根据各个数据的取值情况，可以有下述一种简单的书写格式：plot3(x，y，z)或 plot3(x，y，z，option)。

（3）选项参数 option 指明了所绘图中线条的线性、颜色以及各个数据点的表示记号。

（4）plot3 命令使用的是以逐点连线的方法来绘制三维折线的，当各个数据点的间距较小时，也可利用它来绘制三维曲线。

在 MATLAB 中，除了可以绘制三维曲线以外，还可以绘制三维曲面。常见的绘制三维曲面的 MATLAB 函数有 mesh 和 surf，下面分别介绍这两个函数的用法。

2．三维网格曲面

MATLAB 中可以通过 mesh 函数绘制三维网格曲面图，该函数的常用格式有以下几种：

（1）mesh(***X***,***Y***,***Z***,***C***)

参数 ***X***、***Y***、***Z*** 都为矩阵值，参数 ***C*** 表示网格曲面的颜色分布情况。

（2）mesh(***X***,***Y***,***Z***)

参数 ***X***、***Y***、***Z*** 都为矩阵值，网格曲面的颜色分布与 z 方向上的高度值成正比。

（3）mesh(***x***,***y***,***Z***,***C***)

参数 ***x*** 和 ***y*** 为长度分别是 n 和 m 向量，而参数 ***Z*** 是维数为 $m\times n$ 的矩阵，参数 ***C*** 表示网格曲面的颜色分布情况。

（4）mesh(***x***,***y***,***Z***)

参数 ***x*** 和 ***y*** 为长度分别是 n 和 m 的向量，而参数 ***Z*** 是维数为 $m\times n$ 的矩阵，网格曲面的颜色分布与 z 方向上的高度值成正比。

（5）mesh(***Z***,***C***)

参数 ***Z*** 是维数为 $m\times n$ 的矩阵，参数 ***C*** 表示网格曲面的颜色分布情况。

（6）mesh(***Z***)

参数 ***Z*** 是维数为 $m\times n$ 的矩阵，网格曲面的颜色分布与 z 方向上的高度值成正比。

3．三维阴影曲面

基本的三维阴影曲面绘制采用 surf 函数，调用这种函数的格式有以下几种。

（1）surf(***X***,***Y***,***Z***,***C***)

参数 ***X***、***Y***、***Z*** 都为矩阵值，参数 ***C*** 表示网格曲面的颜色分布情况。

（2）surf(X,Y,Z)

参数 X、Y、Z 都为矩阵值，网格曲面的颜色分布与 z 方向上的高度值成正比。

（3）surf(x,y,Z,C)

参数 x 和 y 为长度分别是 n 和 m 的向量，而参数 Z 是维数为 $m \times n$ 的矩阵，参数 C 表示网格曲面的颜色分布情况。

（4）surf(x,y,Z)

参数 x 和 y 为长度分别是 n 和 m 的向量，而参数 Z 是维数为 $m \times n$ 的矩阵，网格曲面的颜色分布与 z 方向上的高度值成正比。

（5）surf(Z,C)

参数 Z 是维数为 $m \times n$ 的矩阵，参数 C 表示网格曲面的颜色分布情况。

（6）surf(Z)

参数 Z 是维数为 $m \times n$ 的矩阵，网格曲面的颜色分布与 z 方向上的高度值成正比。

在 surf 命令中，各个四边形表面的颜色分布方式可由 shading 命令来指定。

- shading faceted：表示截面式颜色分布方式。
- shading interp：表示插补式颜色分布方式。
- shading flat：表示平面式颜色分布方式。

【例 3-3】 三维曲线绘制函数使用实例。利用 plot3 函数绘制三维螺旋线。其中 $y=\sin t$、$y=\cos t$，$z=t$，$t \in [0,8\pi]$。

解： 在 M 文件编辑器中输入下列程序代码。

```
t=0:pi/50:8*pi;                          %通过 meshgrid 创建网格数据
x=sin(t);    y=cos(t);    z=t;           %定义 x、y、z 与 t 之间的函数关系
plot3(x,y,z)                             %绘制 x、y、z 三维图形
```

执行该程序后，显示结果如图 3-5 所示。

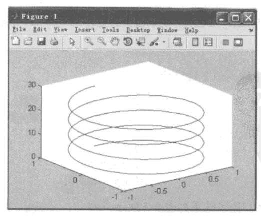


图 3-5　三维螺旋线图形

【例 3-4】 三维网格曲面图绘制应用实例。利用函数 mesh 在笛卡尔坐标系中绘制以下函数的网格曲面图：$f(x,y)=\dfrac{\sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}$。

解：在 M 文件编辑器中输入下列程序代码。

```
x=-8:0.5:8;                    %定义 x 坐标轴范围及刻度
y=x;                           %设置 y 与 x 之间的函数关系
[X,Y]=meshgrid(x,y);           %设置矩形网络
R=sqrt(X.^2+Y.^2)+eps;         %函数关系
Z=sin(R)./R;                   %函数关系
mesh(X,Y,Z)                    %绘制网格曲面
grid on                        %绘制网格
```

运行以上程序，得到函数的三维网格曲面图如图 3-6 所示。

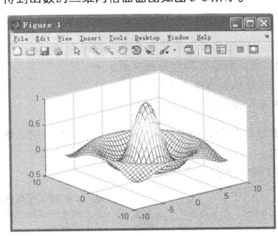


图 3-6 三维网格曲面图

【例 3-5】 阴影曲面绘制实例。利用 surf 函数绘制三维函数 $f(x,y)=\dfrac{2\sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}$ 的三维阴影曲面。

解：在 M 文件编辑器中输入以下程序代码。

```
x=-8:0.5:8;                    %定义 x 坐标轴范围及参数
y=x;                           %定义 y 与 x 之间的函数关系
[X,Y]=meshgrid(x,y);           %设置矩形网络
R=sqrt(X.^2+Y.^2)+eps;         %设置函数关系
Z=2*sin(R)./R;                 %设置函数关系
surf(X,Y,Z)                    %绘制阴影曲面图
```

保存并运行该程序，显示结果如图 3-7 所示。

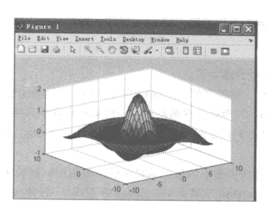


图 3-7 三维阴影曲面

3.4 图形的修饰

图形绘制以后，需要对图形进行标注、说明等修饰性的处理，以增加图的可读性，使之能反映出更多的信息。可以利用 Figure 窗口的菜单和工具栏对图形进行标注、修饰等，操作非常简单，请参考前面的 1.4.6 节。

此外，还可以利用 MATLAB 自带的函数来进行图形的修饰。

1. 选择图形窗口的命令

- figure 命令：打开不同的图形窗口。

figure(1)；figure(2)；…；figure(n)：这个命令用来打开不同的图形窗口，以便绘制不同的图形。

- subplot 命令：拆分图形窗口。

subplot(*m*, *n*, *p*)：分割图形显示窗口，*m* 表示上下分割个数，*n* 表示左右分割个数，*p* 表示子图编号。

2. 坐标轴相关的命令

在默认情况下 MATLAB 自动选择图形的横、纵坐标的比例，当然也可以用 axis 命令控制，常用的命令介绍如下。

- axis([xmin xmax ymin ymax]) ：[xmin xmax ymin ymax]中分别给出 *x* 轴和 *y* 轴的最大值、最小值。
- axis equal：*x* 轴和 *y* 轴的单位长度相同。
- axis square：图框呈方形。
- axis off：清除坐标刻度。

在某些应用中，还会用到半对数坐标轴，MATLAB 中常用的对数坐标绘制命令介绍如下。

- semilogx：绘制以 *x* 轴为对数坐标（以 10 为底）、*y* 轴为线性坐标的半对数坐标图形。
- semilogy：绘制以 *y* 轴为对数坐标（以 10 为底）、*x* 轴为线性坐标的半对数坐标图形。

- loglog：绘制全对数坐标绘图，即 x、y 轴均为对数坐标（以 10 为底）。

3. 文字标示命令

常用的文字标示命令介绍如下。

- text(x, y, '字符串')：在图形的指定坐标位置(x, y)处标示单引号括起来的字符串。
- gtext('说明文字')：利用鼠标在图形的某一位置标示说明文字。执行完绘图命令后再执行 gtext('说明文字') 命令，就可在屏幕上得到一个光标，然后用鼠标选择放置说明文字的位置。
- title('字符串')：在所画图形的最上端显示说明该图形标题的字符串。
- xlabel('字符串')、ylabel('字符串') 、zlabel('字符串')：设置 x、y、z 坐标轴的名称。输入特殊的文字需要用反斜杠（\）开头。
- legend('字符串 1', '字符串 2', …, '字符串 n')：在屏幕上开启一个小视窗，然后依据绘图命令的先后次序，用对应的字符串区分图形上的线。

4. 在图形上添加或删除栅格命令

常用的栅格操作命令介绍如下。

- grid：给图形加上栅格线。
- grid on：给当前坐标系加上栅格线。
- grid off：从当前坐标系中删去栅格线。
- grid：交替转换命令，即执行一次，转变一个状态（相当于 grid on、grid off）。

5. 图形保持或覆盖命令

常用的图形保持和覆盖的命令介绍如下。

- hold on：把当前图形保持在屏幕上不变，同时允许在这个坐标内绘制另外一个图形。
- hold off：使新图覆盖旧图。
- hold：此命令可以保持当前的图形，并且防止删除和修改比例尺。hold 命令是一个交替转换命令，即执行一次，转变一个状态（相当于交替执行 hold on 和 hold off）。



MATLAB 默认 hold 为 hold off，这时的操作会修改图形的属性，因此需要在 plot 之前加上 hold on。

6. 应用型绘图命令

应用型绘图命令常用于数值统计分析或离散数据处理，常用的应用型绘图命令介绍如下。

- box(**x**, **y**)：绘制对应于输入 **x** 和输出 **y** 的高度条形图。
- hist(**y**, **x**)：绘制 **x** 在以 **y** 为中心的区间中分布的个数条形图。
- stairs(**x**, **y**)：绘制 **y** 对应于 **x** 的梯形图。
- stem(**x**, **y**)：绘制 **y** 对应于 **x** 的散点图。



对于图形的属性同样可以在图形窗口上直接编辑，但图形窗口关闭之后编辑结果不会保存。

【例 3-6】 绘图命令使用实例。绘制[0,4π]区间上的 $x1=10\sin t$ 和 $x2=5\cos t$ 曲线，并要求：

（1）$x1$ 曲线的线形为点画线、颜色为红色、数据点标记为加号；$x2$ 曲线为虚线、颜色为蓝色、数据点标记为星号；

（2）标示坐标轴的显示范围和刻度线、添加栅格线；

（3）标注坐标轴名称、标题、相应文本。

解： MATLAB 程序代码如下所示。

```
close all  %关闭打开了的所有图形窗口
clc  %清屏命令
clear    %清除工作空间中所有变量
t=[0:pi/20:4*pi];      %定义时间范围
hold on   %允许在同一坐标系下绘制不同的图形
axis([0 4*pi -10 10]) %横轴范围[0,4π]，纵轴范围[-10,10]
plot(t, 10*sin(t), 'r+:') %线形为点画线、颜色为红色、数据点标记为加号
plot(t, 5*cos(t),'b*--') %线形为虚线、颜色为蓝色、数据点标记为星号
xlabel('时间t'); ylabel('幅值x')    %标注横、纵坐标轴
title('简单绘图实例') %添加图标题
legend('x1=10sint:点画线','x2=5cost:虚线') %添加文字标注
gtext('x1'); gtext('x2') %利用鼠标在图形标示曲线说明文字
grid on    %在所画出的图形坐标中添加栅格，注意用在plot之后
```

运行后，输出结果如图 3-8 所示。

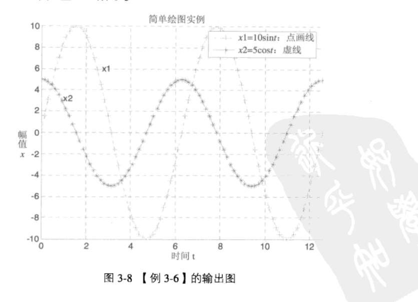


图 3-8 【例 3-6】的输出图

3.5 小结

MATLAB 具有强大的数据可视化功能，可以方便地对数据进行绘图。本章详细讲解了 MATLAB 中图形绘制的流程、函数、工具、图形修饰的方法，以及特殊坐标轴的绘制和多种特殊绘图函数。

本章的例子都只用到了简单的绘图函数和标注函数的组合，都是绘图中最基本最经典的实例，读者都应该仔细阅读体会，最好实践练习。



第 4 章 MATLAB 编程入门

在 MATLAB 中，除了可以在命令窗口中输入命令逐句执行外，也可以和 C、Fortran 等高级语言一样编程，称为 M 文件编程。

读者首先应掌握 MATLAB 程序设计的基本方法，不断实践，才能逐步将其强大的功能应用到科学计算及其他领域的学习和应用中去。

4.1 MATLAB 编程概述

MATLAB 不仅是一种功能强大的高级语言，而且是一个集成的交互式开发环境，用户可以通过 MATLAB 提供的编辑调试器编写和调试 MATLAB 代码。

MATLAB 提供了代码书写和调试的集成开发环境，用户可以在 MATLAB 的代码编辑调试器中完成书写和调试过程。单击 MATLAB 主界面的“新建”工具按钮或者单击“File”（文件）菜单“New”（新建子菜单）的“M-File”项，就可以打开 MATLAB 代码编辑-调试器，其空白界面如图 4-1 所示。

用户也可以在命令窗口通过 edit filename 命令打开已存在的 M 文件进行编辑调试。

从图 4-1 可见，MATLAB 能够根据 M 文件内容区别所打开的是脚本 M 文件还是函数 M 文件，并且在整个编辑过程中追踪光标位置（如图 4-1 底部的“Ln 1 Col 1”表示当前光标处在第一行的第一列），这对于快速准确地定位当前编辑和修改位置是很方便有用的。

开发 MATLAB 程序一般需要经历代码编写、调试、优化几个阶段。

在编写代码时，要及时保存阶段性成果，可以通过“File”菜单的“Save”项或者“保存”按钮保存当前 M 文件。

完成代码书写之后，要试运行代码看看有没有运行错误，然后根据针对性的错误提示对程序进行修改。

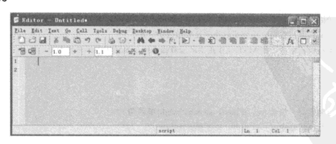


图 4-1 MATLAB 代码编辑-调试器

运行脚本 M 文件，只需要在命令窗口中输入其文件名，然后按回车键，或通过“Debug”

菜单的“Save file and Run”项，或按快捷键“F5”完成。

函数 M 文件需要在命令窗口传递输入参数才能调用。除了一些很简单的代码，大部分情况下用户都可能遇到程序报错的问题，这就需要对代码进行调试纠错，一般需要通过“Debug”菜单下的子项辅助完成，包括设置断点、逐步运行等项。

当程序运行无误后，还要考虑程序性能是否可以改进。MATLAB 提供了 M-Lint 和 Profiler 工具，能够辅助用户分析代码运行中时间消耗的细节和可能需要改变的编程细节，比如循环赋值前没有预定义数组，用循环去实现可以用数组函数实现的运算等。这些工具都在“Tools”菜单下设置了子菜单。

4.2 MATLAB 程序设计原则

MATLAB 程序的基本设计原则如下所述。

（1）百分号“%”后面的内容是程序的注解，要善于运用注解使程序更具可读性。

（2）养成在主程序开头用 clear 指令清除变量的习惯，以消除工作空间中其他变量对程序运行的影响，但注意在子程序中不要用 clear。

（3）参数值要集中放在程序的开始部分，以便维护。要充分利用 MATLAB 工具箱提供的指令来执行所要进行的运算，在语句行之后输入分号使其及中间结果不在屏幕上显示，以提高执行速度。

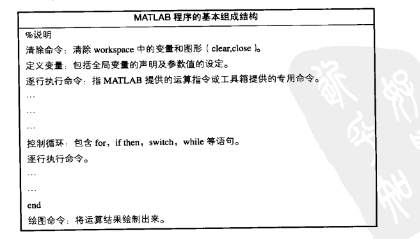
（4）input 指令可以用来输入一些临时的数据；而对于大量参数，则建立一个存储参数的子程序，在主程序中通过子程序的名称来调用。

（5）程序尽量模块化，即采用主程序调用子程序的方法，将所有子程序合并在一起来执行全部的操作。

（6）充分利用 Debugger 来进行程序的调试（设置断点、单步执行、连续执行），并利用其他工具箱或图形用户界面（GUI）的设计技巧，将设计结果集成到一起。

（7）设置好 MATLAB 的工作路径，以便程序运行。

（8）MATLAB 程序的基本组成结构如下所示。



MATLAB 程序的基本组成结构
%说明
清除命令：清除 workspace 中的变量和图形（clear,close）。
定义变量：包括全局变量的声明及参数值的设定。
逐行执行命令：指 MATLAB 提供的运算指令或工具箱提供的专用命令。
…
…
…
控制循环：包含 for，if then，switch，while 等语句。
逐行执行命令。
…
…
end
绘图命令：将运算结果绘制出来。

当然，更复杂的程序还需要调用子程序，或与其他应用程序相结合。

4.3 M 文件

M 文件是包含 MATLAB 代码的文件。

1. M 文件的类型

M 文件按其内容和功能可以分为脚本 M 文件和函数 M 文件两大类。

（1）脚本 M 文件

它是许多 MATLAB 代码按顺序组成的命令序列集合，不接受参数的输入和输出，与 MATLAB 工作空间共享变量空间。

它一般用来实现一个相对独立的功能，比如对某个数据集进行某种分析、绘图，求解某个已知条件下的微分方程等。用户可以通过在命令窗口中直接输入文件名来运行脚本 M 文件。

通过脚本 M 文件，用户可以把为实现一个具体功能的一系列 MATLAB 代码写在一个 M 文件中，每次只需要输入文件名即可运行脚本 M 文件中的所有代码。

（2）函数 M 文件

它也是实现一个单独功能的代码块，但与脚本 M 文件不同的是，函数 M 文件需要接受参数输入和输出，函数 M 文件中的代码一般只处理输入参数传递的数据，并把处理结果作为函数输出参数返回给 MATLAB 工作空间中的指定接收变量。

因此，函数 M 文件具有独立的内部变量空间。在执行函数 M 文件时，要指定输入参数的实际值，而且一般要指定接收输出结果的工作空间变量。

MATLAB 提供的许多函数就是用函数 M 文件编写的，尤其是各种工具箱中的函数，用户可以打开这些 M 文件来查看。实际上，对于特殊应用领域的用户，如果积累了足够多的专业领域应用的函数，就可以组建自己的专业领域工具箱了。

通过函数 M 文件，用户可以把实现一个抽象功能的 MATLAB 代码封装成一个函数接口，在以后的应用中重复调用。

2. M 文件的结构

图 4-2 显示的是 MATLAB 提供的一个函数 M 文件的全部内容，图中清楚地显示了一般的 M 文件包括的各部分结构。

从图 4-2 可以看到，MATLAB 中 M 文件一般包括以下 5 部分结构。

（1）函数声明行（Function Definition Line）。这一行只出现在函数 M 文件的第一行，通过 function 关键字表明此文件是一个函数 M 文件，并指定函数名、输入和输出参数，如图 4-2 中的第 1 行所示。

（2）H1 行。这是帮助文字的第一行（the first help text line），给出 M 文件帮助最关键的信息。当用 lookfor 查找与某个单词相关的函数时，lookfor 只在 H1 行中搜索是否出现指

定单词，如图 4-2 中的第 2 行所示。

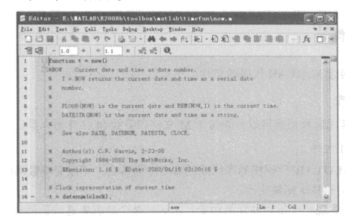


图 4-2 M 文件的一般结构

（3）帮助文字。这部分对 M 文件有更加详细的说明，解释 M 文件实现的功能，M 文件中出现的各变量、参数的意义，以及创作版权信息等，如图 4-2 中的第 13 行所示。当获取一个 M 文件的帮助时，H1 行和帮助文字部分同时显示。

（4）M 文件正文。这是 M 文件实现功能的 MATLAB 代码部分，通常包括运算、赋值等指令。图 4-2 的例子中只有第 16 行，但一般 M 文件正文都由多行代码组成。

（5）注释部分。这部分出现的位置比较灵活，主要是用来注释 M 文件正文的具体运行过程，以方便阅读和修改，经常穿插在 M 文件正文中间。

图 4-2 的例子中的第 15 行就是注释说明正文第 16 行的意义。注释一般都是针对接下来的一段正文代码的，M 文件中也经常包括多行注释。

3．M 文件的创建

虽然一般脚本 M 文件可以包括上述 5 部分结构中除去“函数声明行”以外的 4 部分，但在实际应用中，脚本 M 文件经常仅仅由 M 文件正文和注释部分构成。正文部分实现功能，注释部分则给出每一块代码的功能说明。下面通过实例讲述脚本 M 文件的创建。

【例 4-1】 M 文件创建实例。建立一个命令文件，将变量 a，b 的值互换。

解： 首先打开 M 文件编辑器，输入以下程序。

```
a=1:9;  b=[11,12,13;14,15,16;17,18,19];
c=a;  a=b;  b=c;
a
b
```

然后保存文件名为“41.m”即完成了文件的创建。

在 MATLAB 的命令窗口中输入“41”，将会执行该命令文件。

```
>> 41
a = 11    12    13
    14    15    16
```

```
      17    18    19
b =    1     2     3     4     5     6     7     8     9
```

函数 M 文件的命名一般和函数名一致，比如图 4-2 中函数声明行 function t=now()，表明函数名为 now，因此此函数 M 文件一般保存为 now.m，可以通过 now()语句调用该文件；如果函数名和文件名不一致，函数调用就需要通过文件名和与函数声明中对应的参数列表来实现。

编写好的函数 M 文件，相当于 MATLAB 提供的命令，可以在命令行进行函数调用。但要注意，要求被调用的函数对应的.m 文件必须在 MATLAB 路径下。

4.4 MATLAB 程序流程控制

和各种常见的高级语言一样，MATLAB 也提供了多种经典的程序结构控制语句。MATLAB 中的程序流程控制语句有：分支控制语句（if 结构和 switch 结构）、循环控制语句（for 循环、while 循环、continue 语句和 break 语句）和程序终止语句（return 语句），下面分别进行介绍。

1. 分支控制语句

分支控制语句可以使程序中的一段代码只在满足一定条件时才执行，因此也称为分支选择。MATLAB 中分支语句有两类：if 语句和 switch 语句。

- if 与 else 或 elseif 连用，偏向于是非选择，当某个逻辑条件满足时就执行 if 后的语句，否则执行 else 语句。
- switch 和 case、otherwise 连用，偏向于情况的列举，当表达式结果为某个或某些值时，执行特定 case 指定的语句段，否则执行 otherwise 语句。

if 结构的语法形式如下所示：

```
if logical_expression
   statements
elseif logical_expression
        statements
else    logical_expression
        statements
end
```

其中 elseif 和 else 语句都是可选语句。if、elseif 和 else 构成的各项分支里面，哪个的条件满足（逻辑表达式 logical_expression 的结果为真），就执行哪个分支后面紧跟的程序语句。因此，各个分支条件满足的情况应该是互斥的和完全的，就是被选的条件在一个分支中成立，而且只能在一个分支中成立。当然，省略了 elseif 和 else 分支的语句，就不必要求分支条件满足的情况具备完全性了。

if 结构中条件判断除了可以用逻辑表达式外，还可以用数组 A，这时判断相当于逻辑表达式 all(A)，即当数组 A 的所有元素都为非零值时，才执行该条件后的分支代码。

特别地，当数组 A 为空数组时，相当于该条件判断为假。

switch 结构的语法形式如下所示：

```
switch expression (scalar or string)
   case value1
      statements     % Executes if expression is value1
   case value2
      statements     % Executes if expression is value2
    ...
   otherwise
      statements     % Executes if expression does not
                     % match any case
end
```

执行中，先计算表达式 expression 的值，当结果等于某个 case 的 value 时，就执行该 case 紧随的语句。如果所有 case 的 value 都和 expression 计算结果不相等，则执行 otherwise 后面紧随的语句。

otherwise 语句是可选的，当没有 otherwise 语句时，如果所有 case 的 value 都和 expression 计算结果不相等，则跳过 switch-case 语句段，直接执行后续代码。

相等的意义，对于数值类型来说，相当于判断"if result==value"，对于字符串类型来说，相当于判断"if strcmp(result,value)"。

由此可见，switch-case 语句实际上可以被 if-elseif-else 语句等效替换，不过两种结构各有更便利的地方，读者在以后的例子中会逐渐体会到。

学过 C 语言的读者需要注意，MATLAB 中的 switch-case 结构，只执行表达式结果匹配的第一个 case 分支，然后就跳出 switch-case 结构。因此，不需要在每一个 case 语句中用 break 语句跳出。

在一条 case 语句后可以列举多个值，只需要以元胞数组的形式列举多个值，就是用花括号把由逗号或空格分隔的多个值括起来即可。

2. 循环控制语句

循环控制语句能够使得某段程序代码多次重复执行，MATLAB 中提供了两类循环语句，分别是 for 循环和 while 循环：

- for 循环一般用于已知循环执行次数的情况。
- while 循环则用于已知循环退出条件的情况。

MATLAB 还提供了 continue 和 break 语句，用于更精细地控制循环结构：

- continue 语句使得当前次循环不向下执行，直接进入下一次循环。
- break 语句则表示退出该循环。

（1）for 循环

for 循环用于知道循环次数的情况，其语法格式如下所示：

```
for index = start:increment:end
    statements
end
```

其中，index 为循环变量，increment 为增量，end 用于判断循环是否应该终止。增量 increment 默认值为 1，可以自由设置；当增量为正数时，循环开始先将 index 赋值为 start，然后判断 index 是否小于或等于 end。若是，则执行循环语句，执行完后，对 index 累加一个增量 increment；再判断 index 是否小于或等于 end，若是，则继续执行循环语句，继续对 index 累加，循环往复，直到 index 大于 end 时退出循环。

增量 increment 也可以设置为负整数，表示每次循环执行后对循环变量 index 进行递减，当 index 小于 end 时，退出循环。

MATLAB 中，循环的执行效率很低，提高程序效率的一个办法就是多采用数组结构和 MATLAB 内联函数。

for 循环中的循环变量 index 也可以赋值为数组 $\boldsymbol{A}$，那么在第一次循环中 index 就被赋值为 $\boldsymbol{A}(:,1)$，即 $\boldsymbol{A}$ 的第一列元素，第二次循环 index 被赋值为 $\boldsymbol{A}(:,2)$，以此类推；若 $\boldsymbol{A}$ 有 n 列元素，则循环执行 n 次，第 n 次循环时，循环变量 index 被赋值为 $\boldsymbol{A}(:,n)$。

（2）while 循环

while 循环用于已知循环退出条件的情况，其语法形式如下所示：

```
while expression
   statements
end
```

当表达式 expression 的结果为真时，就执行循环语句，直到表达式 expression 的结果为假，才退出循环。

如果表达式 expression 是一个数组 $\boldsymbol{A}$，则相当于判断 all($\boldsymbol{A}$)。特别地，空数组则被当作逻辑假，循环停止。

（3）continue 语句

continue 语句用在循环中，表示当前循环不再继续向下执行，而是直接对循环变量进行递增，进入下一次循环。

（4）break 语句

break 语句用于退出循环。

3. 程序终止语句

一般程序代码都是按流程执行完毕后正常退出，但当遇到某些特殊情况，程序需要立即退出时，就可以用 return 语句提前终止程序运行。

【例 4-2】 return 语句使用实例。

```
clear
clc
n=-2;
if n<0
   disp('negative number!');
   return;
end
```

```
disp('codon after return')
```

本例中当变量 n 取值为负数时，通过 return 直接退出，不执行 if 后的代码。其运行结果是：

```
negative number!
```

若去掉其中的 return 语句，则运行结果变为：

```
negative number!
codon after return
```

return 语句更多地用在 MATLAB 函数 M 文件中。

4.5 MATLAB 中的函数及调用

4.5.1 函数类型

MATLAB 中的函数可以分为匿名函数、M 文件主函数、嵌套函数、子函数、私有函数和重载函数。

1. 匿名函数

匿名函数通常是很简单的函数，它是面向命令行代码的函数，通常只由一句很简单的声明语句组成。

匿名函数也可以接受多个输入和输出参数。使用匿名函数的优点是不需要维护一个 M 文件，而只需要一句非常简单的语句，就可以在命令窗口或 M 文件中调用函数，这对于那些函数内容非常简单的情况是很方便的。

创建匿名函数的标准格式如下所示：

```
fhandle = @(arglist) expr
```

（1）expr 通常是一个简单的 MATLAB 变量表达式，实现函数的功能，比如 x+x.^2、sin(x).*cos(x)等。

（2）arglist 是参数列表，它指定函数的输入参数列表，对于多个输入参数的情况，通常要用逗号分隔各个参数。

（3）符号“@”是 MATLAB 中创建函数句柄的操作符，表示对由输入参数列表 arglist 和表达式 expr 确定的函数创建句柄，并把这个函数句柄返回给变量 fhandle，这样，以后就可以通过 fhandle 来调用定义好的这个函数。

例如，定义函数：

```
myfunhd=@(x)(x+x..^2)
```

表示创建了一个匿名函数，它有一个输入参数 x，它实现的功能是 x+x.^2，并把这个函数句柄保存在变量 myfunhd 中，以后就可以通过 myfunhd(a)来计算当 x=a 的时候的函数值。

需要注意的是，匿名函数的参数列表 arglist 中可以包含一个参数或多个参数，这样调用的时候就要按顺序给出这些参数的实际取值。

但 arglist 也可以不包含参数，即留空，这种情况下函数时还是需要通过 fhandle()的形式来调用，即要在函数句柄后紧跟一个空的括号。否则，只显示 fhandle 句柄对应的函数形式。

匿名函数可以嵌套，即在 expr 表达式中可以用函数来调用一个匿名函数句柄。

【例 4-3】 匿名函数创建实例。

```
>> myfhd1=@(x)(x+x.^2)
myfhd1 =     @(x)(x+x.^2)
>> myfhd1(2)
ans =     6
>> myfhd2=@(x,y)(sin(x)+cos(y))
myfhd2 =     @(x,y)(sin(x)+cos(y))
>> myfhd2(pi/2,pi/6)
ans =    1.8660
>> myfhd3=@()(3+2)
myfhd3 =     @()(3+2)
>> myfhd3()
ans =     5
>> myfhd3
myfhd3 =     @()(3+2)
>> myffhd=@(a)(quad(@(x)(a.*x.^2+1./a.*x+1./a^2),0,1)) %匿名函数嵌套使用
myffhd =     @(a)(quad(@(x)(a.*x.^2+1./a.*x+1./a^2),0,1))
>> myffhd(0.5)
ans =    5.1667
```

匿名函数可以保存在.mat 文件中，上例中就可以通过“save myfunquad.mat myffhd”把匿名函数句柄 myffhd 保存在 myfunquad.mat 文件中，以后需要用到匿名函数 myffhd 时，只需要使用语句“load myfunquad.mat myffhd”就可以了。

2. M 文件主函数

每一个函数 M 文件第一行定义的函数就是 M 文件主函数，一个 M 文件只能包含一个主函数，并通常习惯上将 M 文件名和 M 文件主函数名设为一致。

M 文件主函数的说法是针对其内部嵌套函数和子函数而言的。一个 M 文件中除了一个主函数外，还可以编写多个嵌套函数或子函数，以便在主函数功能实现中进行调用。

3. 嵌套函数

在一个函数内部，可以定义一个或多个函数，这种定义在其他函数内部的函数就被称为嵌套函数。嵌套可以多层发生，就是说一个函数内部可以嵌套多个函数，这些嵌套函数内部又可以继续嵌套其他函数。

嵌套函数的语法格式如下所示：

```
function x = A(p1, p2)
...
```

```
    function y = B(p3)
    ...
    end
...
end
```

一般函数代码中结尾是不需要专门标明“end”的，但在使用嵌套函数时，无论嵌套函数还是嵌套函数的父函数（直接上一层函数）都要明确标出“end”表示函数结束。

嵌套函数的互相调用需要特别注意，这和嵌套的层次密切相关，如在下面一段代码中：

```
function A(x, y)    %外层函数A（例如主函数）
B(x, y);
D(y);
   function B(x, y) %A的嵌套函数（以A为参照可以称为第一层嵌套函数），B的父函数为A
   C(x);
   D(y);
      function C(x) %B的嵌套函数（以A为参照可以称为第二层嵌套函数），C的父函数为B
      D(x);
      end
   end
   function D(x)    %A的嵌套函数（以A为参照可以称为第一层嵌套函数），D的父函数为A
   E(x);
      function E(x) %D的嵌套函数（以A为参照可以称为第二层嵌套函数），E的父函数为D
      ...
      end
   end
end
```

（1）外层的函数可以调用向内一层直接嵌套的函数（A 可以调用 B 和 D），而不能调用更深层的嵌套函数（A 不可以调用 C 或 E）。

（2）嵌套函数可以调用与自己具有相同父函数的其他同层嵌套函数（B 和 D 可以互相调用）。

（3）嵌套函数也可以调用其父函数或与父函数具有相同父函数的其他嵌套函数（C 可以调用 B 和 D），但不能调用与其父函数具有相同父函数的其他嵌套函数内深层嵌套的函数。

4. 子函数

一个 M 文件只能包含一个主函数，但一个 M 文件中可以包含多个函数，这些编写在主函数后的函数都称为子函数。所有子函数只能被其所在 M 文件中的主函数或其他子函数调用。

所有子函数都有自己独立的声明和帮助、注释等结构，只需要在位置上处在主函数之后即可，而各个子函数的前后顺序都可以任意放置，和被调用的前后顺序无关。

M 文件内部发生函数调用时，MATLAB 首先检查该 M 文件中是否存在相应名称的子函数，然后检查这一 M 文件所在目录的子目录下是否存在同名的私有函数，然后按照 MATLAB 路径检查是否存在同名的 M 文件或内部函数。

根据这一顺序，函数调用时首先查找相应的子函数，因此，可以通过编写同名子函数

的方法实现 M 文件内部的函数重载。

子函数的帮助文件也可以通过 help 命令显示，如 myfun.m 文件中有名为 myfun 的主函数和名为 mysubfun 的子函数，那么可以通过 help myfun>mysubfun 命令来获取子函数 mysubfun 的帮助。

5. 私有函数

私有函数是具有限制性访问权限的函数，它们对应的 M 文件需要保存在名为"private"的文件夹下，这些私有函数代码在编写上和普通的函数没有什么区别，也可以在一个 M 文件中编写一个主函数和多个子函数，以及嵌套函数。

私有函数只能被 private 目录的直接父目录下的脚本 M 文件或 M 文件主函数调用。通过 help 命令获取私有函数的帮助，也需要声明其私有特点，例如要获取私有函数 myprifun 的帮助，就要使用 help private/myprivfun 命令。

6. 重载函数

"重载"是计算机编程中非常重要的概念，它经常用在处理功能类似但参数类型或个数不同的函数编写中。

例如现在要实现一个计算功能，一种情况是输入的几个参数都是双精度浮点类型，同时也有一种情况是，输入的几个参数都是整型变量，这时候，用户就可以编写两个同名函数，一个用来处理双精度浮点类型的输入参数，另一个用来处理整型的输入参数，这样，当用户实际调用函数时，MATLAB 就会根据实际传递的变量类型选择执行其中一个函数。

MATLAB 中重载函数通常放置在不同的文件夹下，通常文件夹名称以符号@开头，然后跟一个代表 MATLAB 数据类型的字符。

例如"@double"目录下的重载函数的输入参数应该是双精度浮点型，而"@int32"目录下的重载函数的输入参数应该是 32 位整型。

4.5.2　函数参数传递

MATLAB 中通过 M 文件编写函数时，只需要指定输入和输出的形式参数列表，只是在函数实际被调用的时候，才把具体的数值提供给函数声明中给出的输入参数。

MATLAB 中参数传递过程是传值传递，也就是说，在函数调用过程中，MATLAB 将传入的实际变量值赋给形式参数指定的变量名，这些变量都存储在函数的变量空间中，这个空间和工作空间变量空间是独立的，每一个函数在调用中都有自己独立的函数变量空间。

例如，编写函数：

```
function y=myfun(x,y,z)
```

在命令窗口通过 a = myfun(3, 2, 0.5)调用此函数，那么 MATLAB 首先会建立 myfun 函数的变量空间，把 3 赋值给 x，把 2 赋值给 y，把 0.5 赋值给 z，然后执行函数实现的代码；在执行完毕后，把 myfun 函数返回的参数 y 的值传递给工作空间变量 a，调用过程结束后，函数变量空间被清除。

1．输入和输出参数的数目

MATLAB 的函数可以具有多个输入或输出参数。通常在调用时，需要给出和函数声明语句中一一对应的输入参数；而输出参数个数可以按参数列表对应指定，也可以不指定。不指定输出参数调用函数时，MATLAB 默认把输出参数列表中第一个参数的值返回给工作空间变量“ans”。

MATLAB 中可以通过 nargin 和 nargout 函数确定函数调用时实际传递的输入和输出参数个数，结合条件分支语句，就可以处理函数调用中指定不同数目的输入/输出参数的情况。

【例 4-4】 显示函数输入和输出参数的数目实例。

```
function [y1,y2]=mytestnio(x1,x2)
if nargin==1
   y1=x1;
   if nargout==2
      y2=x1;
   end
else
   if nargout==1
      y1=x1+x2;
   else
      y1=x1;
      y2=x2;
   end
end
```

这个函数可以处理一个或两个输入参数、一个或两个输出参数的情况。当只有一个输入参数 *x*1 和一个输出参数 *y*1 时，把 *x*1 赋值给 *y*1；当有 1 个输入参数 *x*1 和两个输出参数 *y*1、*y*2 时，把 *x*1 赋值给 y1 和 *y*2；当有两个输入参数 *x*1、*x*2 和一个输出参数 y1 时，把 *x*1+*x*2 的计算结果赋值给 *y*1；当有两个输入参数 *x*1、*x*2 和两个输出参数 *y*1、*y*2 时，把 *x*1 赋值给 *y*1，并把 *x*2 赋值给 *y*2。函数调用结果如下所示：

```
>> x=mytestnio(5)
x =     5
>> [x,y]=mytestnio(5)
x =     5
y =     5
>> mytestnio(5)
ans =     5
>> x=mytestnio(5,7)
x =    12
>> [x,y]=mytestnio(5,7)
x =     5
y =     7
>> mytestnio(5,7)
ans =     5
```

指定了输入和输出参数个数的情况比较好理解，只要对应函数 M 文件中的 if 分支项

即可；而对于不指定输出参数个数的调用情况，MATLAB 是按照指定了所有输出参数的调用格式对函数进行调用的，不过在输出时只是把第一个输出参数对应的变量值赋给工作空间变量 ans。

例如“mytestnio(5,7)”这句函数调用中，实际上是按照“[y1, y2]=mytestnio(x1, x2)”这种形式调用的，在函数变量空间中 x1 被赋值为 5，x2 被赋值为 7，y1 计算结果为 5，y2 计算结果为 7，但函数只把输出参数列表中第一个输出变量（即 y1）的取值返回给工作空间变量 ans，因此，ans 取值为 5。

2. 可变数目的参数传递

函数 nargin 和 nargout 结合条件分支语句，可以处理可能具有不同数目的输入和输出参数的函数调用，但这要求对每一种输入参数数目和输出参数数目的组合分别编写代码。

有些情况下，用户可能并不能确定具体调用中传递的输入参数或输出参数的个数，即具有可变数目的传递参数，MATLAB 中可以通过 varargin 和 varargout 函数实现可变数目的参数传递，使用这两个函数，对于处理具有复杂的输入/输出参数个数组合的情况也是很方便的。

函数 varargin 和 varargout 把实际的函数调用时传递的参数值封装成一个元胞数组，因此，在编写函数实现部分的代码时，就要用访问元胞数组的方法访问封装在 varargin 或 varargout 中的元胞或元胞内的变量。

【例 4-5】 可变数目的参数传递实例。

```
function y=mytestvario(varargin)
temp=0;
for i=1:length(varargin)
    temp=temp+mean(varargin{i}(:));
end
y=temp/length(varargin);
```

本例中的函数 mytestvario 以 varargin 为输入参数，从而可以接受可变数目的输入参数。函数实现部分首先计算了各个输入参数（可能是标量、一维数组或二维数组）的均值，然后计算这些均值的均值。调用结果如下所示：

```
>> mytestvario(4)
ans =     4
>> mytestvario(4,[1 3])
ans =     3
>> mytestvario(4,[1 3],[1 23;23 1],magic(4))
ans =   6.6250
```

对于“mytestvario(4,[1 3],[1 23;23 1],magic(4))”这句函数调用，在函数变量区，varargin 首先被赋值为一个元胞数组“{4,[1 3],[1 23;23 1],magic(4)}”，即 varargin 有 1 行 4 列个元胞，各个元胞中分别存储了一个标量数值、一维行数组、2 行 2 列的二维数组和 4 行 4 列的魔方数组；在函数实现部分，首先创建中间变量 temp，并初始化赋值为零（用来存储各个元胞中数据均值的总和），然后计算每一个元胞中所有数据的均值并将结果累加到 temp

上；最后通过“y=temp/length(varargin)”计算这些均值的均值。

函数 varargin 和 varargout 也可以放置在参数列表中某些必然出现的参数之后，其语法格式有如下几种形式：

（1）function [out1, out2] = example1(*a*, *b*, varargin)，表示函数 example1 可以接受大于或等于两个输入参数，返回两个输出参数；两个必选的输入参数是 *a* 和 *b*，其他更多的输入参数被封装在 varargin 中。

（2）function [*i*, *j*, varargout] = example2(*x*, *y*)，表示函数 example2 接受两个输入参数 *x* 和 *y*，返回大于或等于两个输出参数，前两个输出参数为 *i* 和 *j*，其他更多的输出参数封装在 varargin 中。

函数 varargout 和 varargin 的用法类似，只需要注意访问时按照访问元胞数组的方法，这里就不再举例了。

3．返回被修改的输入参数

MATLAB 函数有独立于 MATLAB 工作空间的自己的变量空间，因此，输入参数在函数内部的修改都只具有和函数变量空间相同的生命期，如果不指定将修改后的输入参数值返回到工作空间，那么在函数调用结束后这些修改后的值将被自动清除。

【例 4-6】 函数内部的输入参数修改实例。

```
function y=mytest(x)
x=x+5;
y=x*2;
```

本例中的 mytest 函数内部，首先修改了输入参数 *x* 的值（$x = x+5$），然后以修改后的 *x* 的值计算输出参数 *y* 的值（$y = x*2$）。调用结果如下所示：

```
>> x=3
x =     3
>> y=mytest(x)
y =    16
>> x
x =     3
```

由此结果可见，调用结束后，函数变量区中的 *x* 在函数调用中被修改，但此修改只在函数变量区有效，这并没有影响到 MATLAB 工作空间变量空间中变量 *x* 的值。函数调用前后，MATLAB 工作空间中的变量 *x* 始终取值为 3。

那么，如果用户希望在函数内部对输入参数所做的修改也对 MATLAB 工作空间的变量有效，就需要在函数输出参数列表中返回此输入参数。

对于本例中的函数，则需要把函数修改为“function [*y*, *x*]=mytest(*x*)”，而在调用时也要使用“[*y*, *x*]= mytest(*x*)”这种形式。

【例 4-7】 函数参数传递实例。将修改后的输入参数返回给 MATLAB 工作空间。

```
function [y,x]=mynewtest(x)
x=x+5;
```

```
y=x*2;
```

MATLAB 工作空间中的调用结果如下所示：

```
>> x=3
x =     3
>> [y,x]=mynewtest(x)
y =    16
x =     8
>> x
x =     8
```

通过本例可见，函数调用后，MATLAB 工作空间中的变量 x 取值从 3 变为 8（3+5），可见通过[y, x]=mynewtest(x)调用，实现了函数对 MATLAB 工作空间变量的修改。

4. 全局变量

通过返回修改后的输入参数，可以保留函数内部对 MATLAB 工作空间变量的修改。而另一种殊途同归的方法则是使用全局变量。声明全局变量需要用到 global 关键词，语法格式为“global variable”。

通过全局变量可以实现 MATLAB 工作空间变量和多个函数变量的共享，这样，多个使用全局变量的函数和 MATLAB 工作空间共同维护这一全局变量，任何一处对全局变量的修改，都会直接改变此全局变量的取值。

在应用全局变量时，通常要在各个函数内部通过 global variable 语句声明，在命令窗口或脚本 M 文件中也要先通过 global 声明，然后进行赋值和调用。

【例 4-8】 全局变量使用实例。

```
function y=myprocess(x)
global T
T=T*2;
y=exp(T)*sin(x);
```

在命令窗口中声明全局变量，然后赋值调用：

```
>> global T
>> T=0.3
T =    0.3000
>> myprocess(pi/2)
ans =    1.8221
>> exp(T)*sin(pi/2)
ans =    1.8221
>> T
T =    0.6000
```

通过本例可见，用 global 将 T 声明为全局变量后，函数内部对 T 的修改也会直接作用到 MATLAB 工作空间中。函数 myprocess 被调用一次后，T 的值从 0.3 变为 0.6（0.3*2）。

4.6 函数句柄

函数句柄实际上提供了一种间接调用函数的方法。创建函数句柄需要用到操作符@。前面已经讲过，匿名函数实际上就是一种函数句柄，而对 MATLAB 提供的各种 M 文件函数和内部函数，也都可以创建函数句柄，从而可以通过函数句柄对这些函数实现间接调用。

函数句柄的优点如下：

（1）方便地实现函数间互相调用。

（2）兼容函数加载的所有方式。

（3）拓宽子函数，包括局部函数的使用范围。

（4）提高函数调用的可靠性。

（5）减少程序设计中的冗余。

（6）提高重复执行的效率。

创建函数句柄的一般语法格式如下所示：

```
fhandle=@function_fileame
```

其中，

- function_filename 是函数所对应的 M 文件的名称或 MATLAB 内部函数的名称；
- “@”是句柄创建操作符；
- fhandle 变量保存这一函数句柄。

例如 fhandle=@sin 就创建了 MATLAB 内部函数 sin 的句柄，并将其保存在 fhandle 变量中，以后就可以通过 fhandle(*x*)来实现 sin(*x*)的功能。

通过函数句柄调用函数时，也需要指定函数的输入参数，比如可以通过 fhandle(arg1, arg2, …, arg*n*)这样的调用格式来调用具有多个输入参数的函数。对于那些没有输入参数的函数，在使用句柄调用时，要在句柄变量后加上空的圆括号，即 fhandle()。

【例 4-9】 函数句柄的创建和调用实例。

```
>> fhd=@sin
fhd =     @sin
>> x=0:0.25*pi:2*pi;
>> fhd(x)
ans = 0    0.7071    1.0000    0.7071    0.0000    -0.7071    -1.0000    -0.7071
-0.0000
```

MATLAB 中提供了丰富的处理函数句柄的函数，如表 4-1 所示。

表 4-1 处理函数句柄的函数

函　数	说　明
functions(fhandle)	返回一个结构体，存储了函数的名称，函数类型（简单函数或重载函数），以及函数 M 文件的位置
func2str(fhandle)	将函数句柄转换为函数名称字符串

续表

函 数	说 明
str2func(str)	将字符串代表的函数转换为函数句柄
save filename.mat fhandle	将函数句柄保存在.mat 文件中
load filename.mat fhandle	把.mat 文件中存储的函数句柄装载到工作空间
isa(var, 'function_handle')	检测变量 var 是不是函数句柄
isequal(fhda, fhdb)	检测两个函数句柄是否对应于同一个函数
feval(fhandle)	调用函数句柄 fhandle

【例 4-10】 处理函数句柄的函数使用实例。

```
>> fhda=@exp
fhda =     @exp
>> fhdb=@myprocess
fhdb =     @myprocess
>> functions(fhdb)
ans =  function: 'myprocess'
       type: 'simple'
       file: 'D:\MATLAB71\work\MATLABbook\EX-10\myprocess.m'
>> isa(fhda,'function_handle')
ans =     1
>> isequal(fhda,fhdb)
ans =     0
```

4.7 MATLAB 程序调试

MATLAB 程序调试主要是指发现和纠正程序中的错误。

4.7.1 常见程序错误

MATLAB 程序常见的错误有以下几类。

1. 矩阵运算方面的错误

（1）矩阵下标索引使用错误

MATLAB 的计算元素是矩阵，即使是一个数，MATLAB 也把它看做一个一维数组。在 MATLAB 中，所有的计算都是以矩阵为单元进行的，矩阵是 MATLAB 的核心。需要非常注意的是：与 C 语言等编程语言的习惯不一样，MATLAB 的语法规定矩阵的索引从 1 开始。因此，在访问矩阵（包括向量、二维矩阵、多维数组）的过程中，下标索引如果从 0 开始，或者出现负数，就会报错。

例如，在命令窗口输入：

```
A=[1,2;2,4];
A(0,1)
```

输出报错为：

```
??? Attempted to access A(0,1); index must be a positive integer or logical.
```

分析：如果改为 **A**(1,1)，则输出为 1，表示访问 **A** 的第 1 行第 1 列的元素。同类的常见错误还有：在引用矩阵元素的时候，索引值超出矩阵应有的范围。

（2）矩阵运算对象维数不匹配的错误

进行矩阵运算时，运算符（=，+，–，/，*等）两边的运算对象维数必须匹配。

例如，在命令窗口输入：

```
A=[1, 2; 3, 3; 4, 5];
B=[1, 2, 3; 4, 5, 6; 7, 8, 9];
A*B
```

输出报错为：

```
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

分析：**A** 是 3*2 的矩阵，**B** 是 3*3 的矩阵，**A*B** 是矩阵维数不匹配，故报错。如果对 **A** 取转置，然后再相乘，则不会报错。

```
>> A'*B
ans =41    49    57
    49    59    69
```

（3）元素与矩阵运算的错误

MATLAB 通过“.”来区分矩阵运算和元素运算，当元素与矩阵进行运算时，容易忽略“.”。

例如，在命令窗口输入：

```
A=[1, 2, 3];
B=6*A    %对 A 的每个元素都乘以 6
C=6/A    %用 6 除以 A 的每个元素
```

输出报错为：

```
B =     6    12    18
??? Error using ==> mrdivide
Matrix dimensions must agree.
```

分析：其实，写为“B=6.*A”进行运算也是正确的，由于习惯的原因，这个“.”通常省略，在作乘法时不会报错，但在作除法时，就错了。改为以下语句就不会出错。

```
C=6./A
C =  6     3     2
```

2. 函数方面的错误

（1）函数没有定义的错误

在命令窗口中可以运行 MATLAB 自带的函数以及用户自己定义的函数，如果不是这两类函数，运行时会报错。

例如，在命令窗口中输入：

```
>> m_fun
```

输出报错为：

```
??? Undefined function or variable 'm_fun'.
```

分析：可能的出错原因有程序文件名错；文件名大小写错；该文件不在搜索路径中。

解决办法：核对文件名；检查大小写，统一命名风格；将该文件复制到或包含在工作路径下。

（2）函数输出变量赋值的错误

在函数中，如果有一个或多个输出变量没有被赋值，调用该函数时，会报错。

分析：函数如果带有输出变量，则每个输出变量在返回的时候都必须被赋值。容易出现这个错误的两个地方是：

- 在部分条件判断语句（如 if ）中没有考虑到输出变量的返回值。
- 在循环迭代过程中部分变量的维数发生了变化。

解决办法：调试程序，仔细查看函数返回时各输出变量的值。更好的方法是，在条件判断或者执行循环之前对所使用的变量赋初值。

（3）在命令窗口定义函数的错误

在 MATLAB 中，不能在命令窗口或者脚本文件中定义函数。例如，在命令窗口输入：

```
>> function c = myfun(a, b)
```

输出报错为：

```
??? function c = myfun(a, b)
Error: Function definitions are not permitted at the prompt or in scripts.
```

分析：在命令窗口写 function c = myfun (a,b)，此错误就会出现，因为函数只能定义在 M 文件中。关于脚本文件和 M 文件的区别请查阅 MATLAB 基础书籍。

简言之：

- 如果写成 function 的形式，那么必须写在 M 文件中，且以 function 开头（即 function 语句前不能包含其他语句，所有语句必须放在 function 中，当然，function 的定义可以有多个，各 function 之间是并列关系，不能嵌套）。
- 如果写成脚本的形式，则既可以写在命令窗口中，也可以写在 M 文件中，但两者均不能包含 function 语句（即不能进行函数的定义）。

解决办法：新建一个 M 文件，然后再进行函数的定义。

总之，对于一些格式错误，如函数名的格式错误、缺括号等，大多数这类错误 MATLAB 可在运行时检测出来，并显示出错信息和位置，编程者也很容易纠正。而算法错误、逻辑上的错误，不易查找，遇到此类错误时需耐心。一般可考虑如下方法：

- 删除句尾分号“;”，注意变量值的变化；将每步执行结果输出到命令窗口，显示中间结果。

- 在适当位置加上 keyboard 语句，当程序执行到这条语句时，MATLAB 会暂停执行，并将控制权交给用户，这时可检查和修改局部工作空间的内容，从中找出错误的线索，利用 return 命令可恢复程序执行。
- 在函数定义行之前加上“%”注释掉，使之变成脚本语言；或者选用“Text”菜单的“Comment”命令，注释掉可疑的代码部分；这样，程序运行出错时便可查看 M 文件中产生的变量。
- 使用 MATLAB 调试器设置断点，或单步执行，使用一些调试和分析工具。

下面讲述程序调试的一些工具及调试方法，熟练掌握并运用这些工具及调试方法，能提高编程的效率。

4.7.2 调试方法

MATLAB 程序有直接调试法和工具调试法这两种调试方法。

1. 直接调试法

直接调试法就是在 M 文件中，将某些语句后面的分号去掉，迫使 M 文件输出一些中间计算结果，以便发现可能的错误。常用的做法有：

（1）在适当位置，添加显示某些关键变量值的语句。

（2）利用 echo 指令，使文件运行时在屏幕上逐行显示文件内容，echo on 能显示 M 脚本文件；echo FunName On 能显示名为 FunName 的 M 函数文件。

（3）在原 M 脚本或函数文件的适当位置，添加指令 keyboard，利用该语句可以设置程序的断点。

（4）通过将原 M 函数文件的函数声明行注释掉，可使一个中间变量难于观察的 M 函数文件变为一个所有变量都保存在基本工作空间中的 M 脚本文件。

2. 工具调试法

工具调试法就是在程序中设置一些断点，利用调试菜单（Debug）中的一些选项进行调试。

MATLAB 提供了进行代码调试和代码分析优化的工具，对这些工具一般的 MATLAB 用户都应该有所了解。尤其是断点调试部分的内容，建议读者尽量以自己的程序代码为例，多加练习，熟练掌握。

4.7.3 调试工具

当完成 MATLAB 代码编写后，用户就可以在命令窗口中运行代码（脚本或函数文件）。对于比较简单的代码，一般只要编程习惯较好，都可以一次通过。但对于很多比较复杂的情况，或者用户初学 MATLAB 编程，就容易在运行时出现错误。这时候，就需要利用 MATLAB 的调试工具对出现错误的代码进行调试纠错。

MATLAB 的代码编辑-调试器是一个综合了代码编写、调试的集成开发环境。MATLAB

代码调试过程，主要是通过 MATLAB 代码编辑-调试器的 Debug 菜单下的子项进行的，如图 4-3 所示。



图 4-3 MATLAB 代码编辑-调试器的 Debug 菜单

Debug 菜单用于程序调试，需要与 Breakpoints 菜单项配合使用。MATLAB R2008b 的 Debug 菜单中的菜单项介绍如下。

（1）Open M-Files when Debugging：用于调试时打开 M 文件。

（2）Step：在调试模式下，执行 M 文件的当前行，对应的快捷键是 F10。

（3）Step In：在调试模式下，执行 M 文件的当前行，如果 M 文件当前行调用了另一个函数，那么进入该函数内部，对应的快捷键是 F11。

（4）Step Out：当在调试模式下执行 “Step In” 进入某个函数内部之后，执行 “Step Out” 可以完成函数剩余部分的所有代码，并退出函数，暂停在进入函数内部前的 M 文件所在行末尾。

（5）Save File and Run：运行当前 M 文件，快捷键是 F5；当前 M 文件设置了断点时，运行到断点处暂停。

（6）Run configuration for：运行调试配置文件。打开需调试的 M 文件后，单击工具栏的 按钮，将弹出一个如图 4-4 所示的该 M 文件的运行配置文件的编辑窗口。

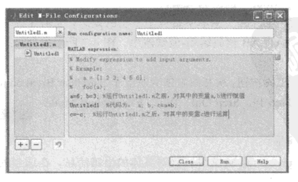


图 4-4 运行配置文件的编辑窗口

在该编辑窗口，用户可以添加一些便于调试的代码、变量赋值、输入参数、中间变量结果等。

在图 4-4 所示的例子中，在该配置文件中，在 M 文件运行之前对其中的参数进行了赋值，运行之后对其中的参数进行了运算。

有了该配置文件后，程序的运行结果 $c=-18$。读者可自己体验一下，加强掌握。

（7）Go Until Cursor：运行当前 M 文件到在光标所在行的行尾。

需要注意，以上这些调试项，除了“Run”（运行），都需要首先在 M 文件中设置断点，然后运行到断点位置后，这些调试项才可启用。

（8）Set/Clear Breakpoint：在光标所在行开头设置或清除断点。

（9）Set/Modify Conditional Breakpoint…：在光标所在行的开头设置或修改条件断点，选择此子项，会打开“条件断点设置”对话框，如图 4-5 所示。此对话框用于设置在满足什么条件时，此处断点有效。

（10）Enable/Disable Breakpoint：将当前行的断点设置为有效或无效。

（11）Clear Breakpoints in All Files：清除所有 M 文件中的断点。

（12）Stop if Errors/Warnings…：设置出现某种运行错误或警告时，停止程序运行，选择此项，会打开“错误/警告设置”对话框，如图 4-6 所示。

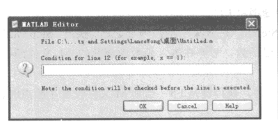


图 4-5 “条件断点设置”对话框

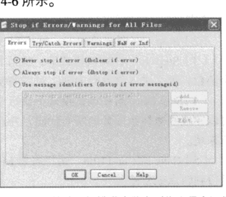


图 4-6 设置当出现某种运行错误或警告则停止程序运行

（13）Exit Debug Mode：退出调试模式。

上面逐项讲述了“Debug”菜单下每一个子项的意义，实际上，很多子项都有对应的快捷工具按钮。在 MATLAB 代码编辑-调试器中，图 4-7 所示的部分工具按钮就是用于 M 文件调试的。

图 4-7 中的各个工具按钮，从左向右依次对应于 Set/Clear Breakpoint、Clear Breakpoints in All Files、Step、Step In、Step Out、Run、Exit Debug Mode 等菜单子项。

图 4-7 调试工具按钮

通常的调试步骤是：

① 先运行（Run）一遍 M 文件，针对具体的出错信息，在适当的地方设置断点或条件断点。

② 再次运行（Run）到断点位置（如图 4-8 所示），此时 MATLAB 把运行控制权交给键盘。

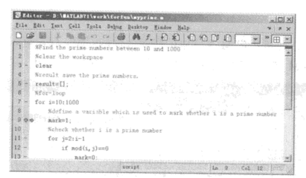


图 4-8　设置断点后运行（Run）到断点所在位置

③ 此时命令窗口出现“K>>”提示符（如图 4-9 所示），在命令窗口中查询 M 文件运行过程中的所有变量，包括函数运行时的中间变量。

④ 运行到断点位置后，用户可以选择“Step/Step Into/Step Out”等调试运行方式，逐行运行并适时查询变量取值，从而逐渐找到错误所在行并排除错误。

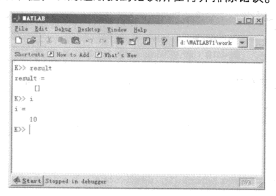


图 4-9　调试模式时 MATLAB 命令窗口把控制权交给键盘

4.7.4　M 文件分析工具

通过“Debug”菜单对 M 文件进行调试，可以找出文件中的编写错误和运行错误并纠正。完成了调试过程后，用户编写的 M 文件就可以正确地运行了。但可能运行效率还不是最优，这就需要通过 MATLAB 提供的分析工具对代码进行分析，然后有针对性地进行优化。

MATLAB 7 的新功能中，有一项是可以检验所选写程序的执行效率。这个功能位于编辑器的工具的一个选项中，称为 M-LINT 检验程序代码（M-Lint）。

这个选项还可以在程序完成调试后，再进一步检查程序的执行效率，生成一个建议的报告（M-Lint Code Check Report），其内容包括相关修正事项，以及如何使程序执行时更有效率，且节省处理时间等。对于大程序而言，这项功能相当有用。

MATLAB 提供的 M 文件分析工具包括 M-Lint 工具和 Profiler 工具，它们都有图形操作界面，使用简单方便，是 MATLAB 程序分析优化的必用工具。

M-Lint 分析工具

M-Lint 工具可以分析用户 M 文件中的错误或性能问题。用户可以先在代码编辑-调试器中打开待分析的 M 文件，然后选择"Tools"菜单下的"M-Lint"项，如图 4-10 所示。

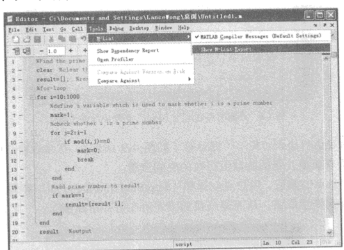


图 4-10　通过 Tools 菜单打开 M-Lint 工具

图 4-10 所示的是实现查找 10～1000 之内所有素数的 M 文件，它的脚本如下。

```
%Find the prime numbers between 10 and 1000
clear  %clear the workspace
result=[];  %result save the prime numbers.
%for-loop
for i=10:1000
    %define a variable which is used to mark whether i is a prime number
    mark=1;
    %check whether i is a prime number
    for j=2:i-1
        if mod(i,j)==0
            mark=0;
            break
        end
    end
    %add prime number to result
    if mark==1
        result=[result i];
    end
end
result   %output
```

运行 M-Lint 工具后结果如图 4-11 所示。

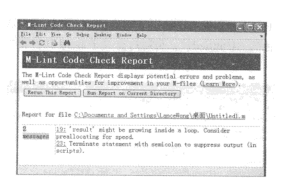


图 4-11　M-Lint 分析结果

从图 4-11 可以看出，M-Lint 分析完成后，返回一个浏览器界面的分析报告（Check Report），报告中包括被分析的 M 文件的路径，以及若干个分析结果（图 4-11 所示中的“2 messages”表示只有 2 条分析结果）。

M-Lint 分析结果中经常出现的错误或问题报告包括：没有以分号结束以阻止中间变量输出，变量在文件中从没有被其他语句调用，以及循环过程中数组尺寸会增加等。

实际上，M-Lint 分析得到的问题，并不一定必须消除，而是要具体问题具体分析。当用户认可某一条分析结果时，单击分析结果中的行号，就可以打开相应的 M 文件并定位到该行，方便地修改代码了。

M-Lint 不仅可以分析单个 M 文件，还可以分析一个文件夹下的所有 M 文件。通常在 MATLAB 主界面下，选择“Desktop”菜单下的“Current Directory”，则可以显示文件夹面板，通过单击此面板顶部的 M-Lint 工具则可以分析相应文件夹下的所有 M 文件。

4.7.5　Profiler 分析工具

Profiler 工具是 MATLAB 提供的另一个功能强大的代码分析工具，利用它可以获取每行代码的运行情况，包括运行时间和调用次数等，因而知道哪些语句行花费的时间最多，可以集中精力进行改进和优化。

使用 Profiler 时，用户可以提前在代码编辑调试器中打开 M 文件，然后选择“Tools”菜单下的“Open Profiler”项，就打开了 Profiler 的图形界面，运行 Profiler 分析工具。

单击图形界面中所示的“Start Profiling”按钮，就可以分析此 M 文件，分析结果如图 4-12 所示。

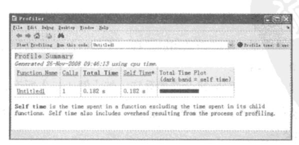


图 4-12　Profiler 分析结果

从图 4-12 可见，Profiler 分析结果给出了调用函数名称、调用次数、消耗总时间等信息。单击图 4-12 中蓝色的“Untitled1”，可以打开关于此 M 文件更加详细的分析报告，如图 4-13 所示。

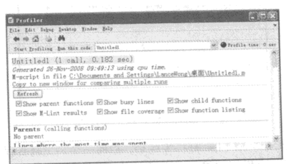


图 4-13 更详细的 Profiler 分析结果

分析报告可以根据用户的需要显示所需的内容，图 4-13 中有 6 项显示内容可供选择。图 4-13 所示的更详细的 Profiler 分析结果提供了 Untitled1.m 文件运行中最消耗时间的部分及其具体耗时信息。用户可以有针对性地修改那些最消耗时间的代码。

一般来说，应该尽量避免不必要的变量输出，循环赋值前要预定义数组尺寸，多采用向量化的 MATLAB 函数，少采用数组，这些都能够提高 MATLAB 程序的运行性能。

4.8 MATLAB 程序设计技巧

MATLAB 是一种科学计算语言，但同时也具有和 C、Fortran 等高级语言相类似的语言特征，能方便地实现程序控制。利用 MATLAB 的程序控制功能，可以将有关 MATLAB 命令编成程序存储在一个文件中（M 文件），然后运行该文件，MATLAB 就会自动依次执行文件中的命令，直到全部命令执行完毕。

编程时，首先要考虑到变量初值或者变量类型改变时，程序的应对能力，即程序的鲁棒性，因此出色的程序要具有较好的例外处理机制。此外，还要考虑程序的执行效率。

一些编程技巧能提高程序的执行效率，因此，掌握一些 MATLAB 的编程技巧也是非常必要的，下面对常用的 MATLAB 编程技巧进行介绍。

4.8.1 嵌套计算

一个程序的执行速度取决于它所调用的子程序个数以及采用的算法。通常希望子程序越少越好，算法效率越高越好。

嵌套计算是一种具有较小时间复杂度的算法，如【例 4-11】所示。

【例 4-11】 嵌套计算与直接求值的比较实例。考虑下面两个多项式，其中式（**）为式（*）的嵌套表达式：

$$p(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \qquad (*)$$

$$p1(x) = ((a_3x + a_2)x + a_1)x + a_0 \qquad (**)$$

解：易知式（*）和式（**）所需要的乘法数分别为：6 次和 3 次。显然后者具有更高的执行效率。

下面用具体程序进行说明，创建函数 ex0411。

```
function ex0411()
N = 100000;
a = [1:N];
x = 1;
tic                                     %初始化时钟
p1 = sum(a.*x.^[N-1:-1:0]);             %按照(*)求值
p1, toc                                 %时钟停止，获得执行时间
tic, p2=a(1);
for i = 2:N                             %嵌套计算(**)
    p2 = p2*x + a(i);
end
p2, toc
tic, p3 = polyval(a, x), toc            %用 MATLAB 自带函数求值
```

运行函数，输出结果如下所示：

```
p1 =  5.0001e+009
Elapsed time is 0.035852 seconds.
p2 =  5.0001e+009
Elapsed time is 0.005543 seconds.
p3 =  5.0001e+009
Elapsed time is 0.088367 seconds.
```

可见，嵌套算法耗时最少，而用 polyval 求值执行速度最慢。

采用嵌套算法不仅能够提高执行效率，而且此方法具有更强的解决问题的能力，下面的实例可以说明这个问题。

【例 4-12】 嵌套计算与非嵌套计算的比较实例。求 possion 分布的有限项和，形如：

$$S(M) = \sum_{n=0}^{M} \frac{\lambda^n}{n!} e^{-\lambda}$$

解：由概率论知识可知，当 M 很大时，上式的值趋近于 1。

分别用式（*）和式（**）来求解，创建函数 ex0412。

```
function ex0412()
r = 80;
M = 160;
p = exp(-r);
S1 = 0;
for k = 1:M
    p=p*r/k;                    %嵌套,式(**)
```



```
    S1=S1+p;
end
S1
S2= 0;
for k = 1:M
    p = r^k/factorial(k);          %非嵌套,式(*)
    S2 = S2 + p;
end
S2*exp(-r);
S2
```

运行函数，输出结果如下所示：

```
S1 =    1.0000
S2 =  5.5406e+034
```

由结果可知，嵌套方法的结果非常接近真实值，而非嵌套的方法根本无法得到正确的结果。

4.8.2 循环计算

循环计算可按照给定的条件，重复执行指定的语句。MATLAB 用于实现循环计算的语句有 for 语句和 while 语句。

对于循环的使用需要注意以下几点。

（1）尽量避免使用循环。在 MATLAB 编程中，采用循环会降低程序的执行速度，应尽量避免使用，可以用其他方式，如向量运算等代替。

（2）为了得到最大的速度，在 for 循环被执行之前，应预先分配数组。否则，在 for 循环内每执行一次命令，就会对数组重新分配一次内存，这样会降低 MATLAB 的执行效率。

（3）优先考虑内联（inline）函数，矩阵运算应该尽量采用 MATLAB 的内联函数，因为内联函数是由更底层的编程语言 C 语言构造的，其执行速度显然快于使用循环的矩阵运算。

（4）应用 MEX 技术。尽管采用了很多措施，但执行速度仍然很慢，比如耗时的循环是不可避免的，这样就应该考虑用其他语言，如 C 或 Fortran 语言。按照 MEX 技术要求的格式编写相应部分的程序，然后通过编译连接，形成在 MATLAB 中可以直接调用的动态连接库（DLL）文件，这样可以显著地加快运算速度。有关 MEX 技术及其应用的详细内容可参考相关书籍。

4.8.3 使用例外处理机制

优秀的程序员能够指导用户如何使用他编写的程序，而且在用户使用不当时，能够给出错误提示信息，并引导用户正确使用函数。前面编写的一些程序都没有相应的错误处理机制，如函数 ex0413，其输入应大于零，当输入小于零时会得到错误的结果。【例 4-13】给出了这样的实例。

【例 4-13】 例外处理机制使用实例。用户输入错误的函数参数时的例外处理，给函

数 ex0413 输入错误的参数，如下所示：

```
>> ex0413(-1)
Elapsed time is 0.000017 seconds.
m =     1
```

显然错误的结果是由于用户不清楚函数参数的取值范围导致的。这种错误由于并不影响程序的运行，因而很难被发现。所以程序员在编程时，应当考虑到可能发生的此类错误，并给出处理错误的机制和错误提示信息。

对上例，如果用户输入了错误的参数，可以采用下面语句终止程序，并提示出错，

```
if n<0
   error('input must be positive, stopped');
end
```

此时，再执行上面的命令，结果如下所示：

```
>> ex0413 (-1)
??? Error using ==> ex0413
input must be positive, stopped
```

这种错误多数都是由于越界造成的，尤其在使用矩阵时，要注意引用矩阵位置不要超过它的边界。另外，如果用户输入的函数参数超过设定的最大个数，或者类型不合要求也会出现这种错误。

但一般而言，对于输入参数小于设定个数的情形，MATLAB 内置程序一般会对未赋值参数作默认处理。典型的例子是 plot(***Y***, ***X***)函数，plot(***Y***)默认的 ***X*** 坐标是[0, 1, 2…]序列。

程序员在编写程序的时候也应当注意处理这种情况。采用 nargin 函数可以判断输入参量的个数，从而设定未被指定的输入参数的值或者直接报错。

【例 4-14】　nargin 函数应用实例。利用 nargin 函数，实现两个多项式的相加，并具有一定的报错功能。

解： 编写函数 ex0414。

```
function p=ex0414(a, b)         %求多项式 a, b 的和 p
if nargin==1                     %输入参数个数为 1，另一个默认参数为全 0 向量
   b=zeros(4, 1);
elseif nargin==0
   error('empty input');      %输入参数个数为 0，报错
end
a=a(:).' ;b=b(:).' ;
na=length(a) ; nb=length(b) ;
p=[zeros(1, nb-na) a]+[zeros(1, na-nb) b];   %多项式相加
```

在 MATLAB 命令窗口中输入正确的参数，调用该函数，输出正确的结果，如下所示：

```
>> a=[1 2 3 4];
>> ex0414(a)
ans =     1     2     3     4
```

在 MATLAB 命令窗口中输入错误的参数，调用该函数，输出报错结果，如下所示：

```
>> ex0414()
??? Error using ==> ex0414
empty input
```

可见，用户输入参数不合要求时，会作默认处理或报错，使得程序具有更强的适应性。

4.8.4 使用全局变量

全局变量是指在不同的工作空间以及基本的工作空间中可以共享的变量。用户只需要在主程序或者任何子程序中声明一个或多个全局变量，则在函数和主程序中都可以直接引用它们，采用如下格式生成全局变量：

```
global v1 v2…vn
```

表达式中各变量之间用空格隔开。

使用全局变量时要注意以下几点：

（1）利用全局变量在主程序和函数之间不需要经过输入或输出变量就可以直接传递数据。但要注意在函数调用中使用它们时，调用结束后全局变量在工作空间中仍然存在。

（2）两个或多个函数也可以共有一个全局变量，只要同时在这些函数中用 global 语句定义即可。

（3）使用全局变量时必须十分小心，最好把全局变量名取得长一些或全部用大写，以免与函数中的局部变量重名。如果重名，容易出现致命错误。所以，使用全局变量不是一个好的编程方法。

（4）一旦变量被声明为全局的，则在任何声明它的地方都可以对它进行修改。这在一定程度上破坏了子程序的独立性。如果全局变量被多个子程序修改，则用户很难知道全局变量的确切值，这使得程序的可读性大大下降。

下面用实例进行说明全局变量的用法。

【例 4-15】 全局变量使用实例。本例用以说明全局变量的声明及函数传递。

建立子程序 ex0415.m 和主程序 ex0415main.m，同时在子程序 ex0415.m 以及主程序 ex0415main.m 中定义全局变量 *D*，具体程序如下所示：

```
function x = ex0415(t, D)
global D                        %声明全局变量
t(find(t == 0)) = eps;          %防止分母出现 0 项
x = sin(pi*t/D)./(pi*t/D);
function ex0415main()%主函数
global D
D = 2;
b1 = -2;
b2 = 2;
t = b1 +[0:100]/100*(b2 - b1);
%通过全局变量传递参数
```



```
plot(t, ex0415(t))
```

本程序运行结果如图 4-14 所示。

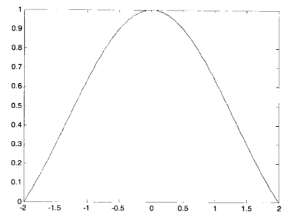


图 4-14 【例 4-15】输出结果

如果在子程序 ex0415.m 中修改全局变量的值，则变量声明时即对其进行赋值。例如将 ex0415.m 中的声明语句改为：global *D*=1，则运行后可得图 4-15 所示结果。

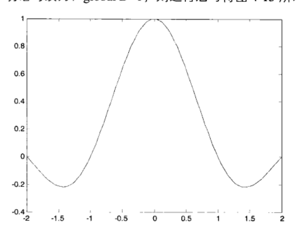


图 4-15 【例 4-15】中全局变量在子程序中被修改的输出结果

显然，ex0415.m 中对全局变量的赋值覆盖了 ex0415main.m 中的赋值，但这种修改在主程序中却不能得到反映。如果主程序和子程序不是同一个程序员编写的，几乎很难找到结果与程序不符合的原因。不过，在使用全局变量时，MATLAB 会给出如下的警告信息，提示用户注意全局变量是否在其他地方被修改：

```
Warning: The value of local variables may have been changed to match the
globals. Future versions of MATLAB will require that you declare a variable to
be global before you use that variable.
```

4.8.5 通过 varargin 传递参数

在编写函数时 varargin 只能作为函数的最后一个参数，主要传递函数中调用的子函数中可选项的参数，其大小也随着输入参数的变化而发生改变。

【例 4-16】 通过 varargin 传递参数的实例。该实例实现在不同输入参数时对函数 ex0415 的作图，其中通过 varargin 传递可选参数。

解： 分别编写主程序 ex0416.m 和作图程序 ex0416plot.m。

```
function ex0416()  %主程序。通过 varargin 传递可选参数
D = 1; b1 = -2; b2 = 2;
t = b1+[0:100]/100*(b2 - b1);
bounds = [b1 b2];
subplot(1,3,1),  ex0416plot('ex0415')    %作出 x 的函数图，bouds 为默认值[-1,1]
axis([b1 b2 -0.4 1.2])
subplot(1,3,2),  ex0416plot('ex0415',bounds)%输入两参数，作[-2,2]上 x 函数图
axis([b1 b2 -0.4 1.2])
subplot(1,3,3), ex0416plot('ex0415',bounds,D)  %可选项输入为 1
axis([b1 b2 -0.4 1.2])
function ex0416plot(ftn,bounds,varargin) %子程序。varagin 为可选变量，输入时
                                          %可以不考虑
if nargin < 2
   bounds = [-1 1];           %bounds 默认为[-1,1]
end
b1 = bounds(1); b2 = bounds(2);
t = b1 + [0:100]/100*(b2 - b1);
x = feval(ftn, t, varargin{:});
plot(t,x)
```

运行主程序，输出结果如图 4-16 所示。

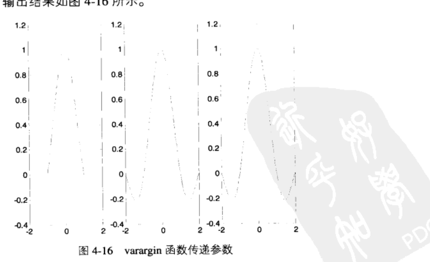


图 4-16 varargin 函数传递参数

对于一个具有多输入参数的程序，传统的方法是针对所有可能出现的输入情况进行处理。上例对应了 3 种输入参数情况，即输入参数个数分别为 1、2、3 的情况。普通处理方法是利用 if-elseif-else 结构对其进行处理。当变量很多时，该方法过于烦琐。

由于 varargin 本身可以根据输入参数个数的变化自动调节大小，因此，在调用具有多个可选参数的子程序的过程中使用 varargin 时，可以大大简化程序。

4.9 小结

本章围绕数值计算介绍了 MATLAB 程序设计的基础知识，包括 MATLAB 的基本操作和编程技巧，这些都是后面内容的基础。

MATLAB 拥有众多的内置函数，学习 MATLAB 时，读者不要试图完全记住或者掌握它们，需要学会使用 help、lookfor 等命令查找所需的命令或函数。

在编写 MATLAB 程序，尤其是大型、复杂的 MATLAB 程序时，要多从用户角度考虑，力求让程序的例外处理机制更完善，具有更好的可读性，但同时也要考虑算法的执行效率，找到这两方面的一个较好的平衡。



第 5 章 Simulink 仿真入门

Simulink 的出现给控制系统分析与设计带来了福音。它有两个主要功能：Simu（仿真）和 Link（连接），即该软件可以利用鼠标在模型窗口上绘制出所需要的控制系统模型，然后利用 Simulink 提供的功能来对系统进行仿真和分析。

在实际工程中，控制系统的结构往往很复杂，如果不借助专用的系统建模软件，则很难准确地把一个控制系统的复杂模型输入计算机，对其进行进一步的分析与仿真，可见，掌握 Simulink 对于一个从事自动控制方面工作的人来说是非常必要的。

5.1 Simulink 仿真概述

Simulink 是 MATLAB 软件的扩展，它是实现动态系统建模和仿真的一个软件包。它与 MATLAB 语言的主要区别在于，它与用户交互接口是基于 Windows 的模型化图形输入的，从而使得用户可以把更多的精力投入到系统模型的构建而非语言的编程上。

所谓模型化图形输入是指 Simulink 提供了一些按功能分类的基本系统模块，用户只需要知道这些模块的输入、输出及模块的功能，而不必考察模块内部是如何实现的，通过对这些基本模块的调用，再将它们连接起来就可以构成所需要的系统模型（以.mdl 文件进行存取），进而进行仿真与分析。

Simulink 的最新版本是 Simulink 7.4（包含在 MATLAB R2009b 里），MATLAB R2007b 里的版本为 7.0 版，它们的基本功能相差不大。

5.1.1 Simulink 的启动与退出

Simulink 的启动有两种方式。一种是启动 MATLAB 后，单击 MATLAB 主窗口的快捷按钮 来打开 “Simulink Library Browser” 窗口，如图 5-1 所示。另一种是在 MATLAB 命令窗口中输入 “Simulink”，在桌面上会出现一个叫做 “Simulink Library Browser” 的窗口，在这个窗口中列出了按功能分类的各种模块的名称。

在 MATLAB 命令窗口中输入 “simulink3”，在桌面上会出现一个用图标形式显示的 “Library :simulink3” 的 Simulink 模块库窗口，如图 5-2 所示。这两种模块库窗口界面只是显示形式不同，用户可以根据个人喜好选用，一般第二种窗口更直观、形象，适合初学者，但使用时会打开太多的子窗口。

Simulink 启动后，便可打开如图 5-3 所示的 Simulink 的仿真编辑窗口，用户此时就可以开始编辑自己的仿真程序了。

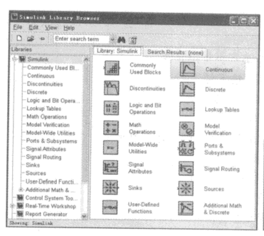


图 5-1 Simulink 模块库浏览界面

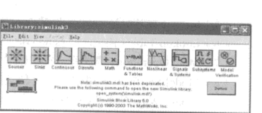


图 5-2 Simulink 模块库窗口

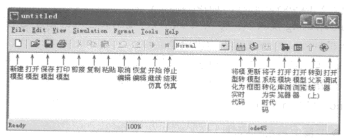


图 5-3 Simulink 仿真编辑窗口

Simulink 的退出操作比较简单，只要关闭所有模型窗口和 Simulink 模块库窗口即可。

5.1.2 Simulink 模块库

在进行系统动态仿真之前，应绘制仿真系统框图，并确定仿真所需用的参数。Simulink 模块库包含大部分常用的建立系统框图的模块，如图 5-4 所示。下面简要介绍常用模块。

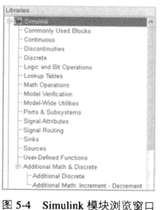


图 5-4 Simulink 模块浏览窗口

1. Simulink 模块库分类

Simulink 模块库按功能分为以下 16 类子模块库。

（1）Commonly Used Blocks 模块库：为仿真提供常用元件。

（2）Continuous 模块库：为仿真提供连续系统元件。

（3）Discontinuities 模块库：为仿真提供非连续系统元件。

（4）Discrete 模块库：为仿真提供离散元件。

（5）Logic and Bit Operations 模块库：提供逻辑运算和位运算的元件。

（6）Lookup Tables 模块库：线形插值查表模块库。

（7）Math Operations 模块库：提供数学运算功能元件。

（8）Model Verification 模块库：模型验证库。

（9）Model-Wide Utilities 模块库：进行模型扩充。

（10）Ports & Subsystems 模块库：端口和子系统。

（11）Signals Attributes 模块库：信号属性模块。

（12）Signals Routing 模块库：提供用于输入、输出和控制的相关信号及相关处理。

（13）Sinks 模块库：为仿真提供输出设备元件。

（14）Sources 模块库：为仿真提供各种信号源。

（15）User-defined Functions 模块库：用户自定义函数元件。

（16）Additional Math & Discrete 模块库：附加的数学和离散模块库。

2. 过程控制系统仿真中常用的模块

Simulink 的模块库能进行很多领域的仿真，如控制系统、通信系统等。下面对过程控制系统仿真中经常用到的模块进行简要介绍，其他模块的使用方法其实都类似，具体的细节请参考 Simulink 自带的帮助文档。

（1）信号源部分模块

过程控制系统仿真中，在信号源部分常用的是输入源模块（Sources），其中常用的子模块如表 5-1 所示。

表 5-1　常用的输入源模块

图　标	模 块 名	功　能
	Band-Limited White Noise	带限白噪声
	Clock	显示和提供仿真时间
1	Constant	产生一个常值
untitled.mat	From File(.mat)	从文件读取数据
simin	From Workspace	从工作空间中读取数据
1	In1	输入信号

续表

图　标	模 块 名	功　能
	Pulse Generator	脉冲发生器
	Ramp	斜坡输入
Signal 1	Signal Builder	信号创建器
	Signal Generator	产生各种不同的波形
	Sine Wave	产生一个正弦波信号
	Step	产生一个阶跃信号
	Uniform Random Number	产生一致随机数

（2）连续模块

过程控制系统仿真中，连续模块（Continuous）中常用的子模块如表 5-2 所示。

表 5-2　常用的连续模块

图　标	模 块 名	功　能
du/dt	Derivative	输入信号微分
$\frac{1}{s}$	Integrator	输入信号积分
$x' = Ax+Bu$ $y = Cx+Du$	State-Space	状态空间系统模型
$\frac{1}{s+1}$	Transfer Fcn	传递函数模型
	Transport Delay	固定时间传输延迟
	Variable Transport Delay	可变时间传输延迟
$\frac{(s-1)}{s(s+1)}$	Zero-Pole	零极点模型

（3）数学运算模块

过程控制系统仿真中，数学运算模块（Math Operations）中常用的子模块如表 5-3 所示。

表 5-3　常用的数学运算模块

图　标	模 块 名	功　能
|u|	Abs	取绝对值
+ +	Add	加法
× ÷	Divide	对输入信号进行除法运算
•	Dot Product	对输入信号进行点乘运算
1	Gain	对输入信号乘以一个常数增益
e^u	Math Function	包括指数函数、对数函数、求平方、开根号等常用数学函数
min	MinMax	最值运算
×	Product	乘运算
+ -	Subtract	减法
++	Sum	求和运算

（4）非连续模块

过程控制系统仿真中，非连续模块（Discontinuous）中常用的子模块如表 5-4 所示。

表 5-4　常用的非连续模块

图　标	模 块 名	功　能
	Backlash	间隙非线性
	Coulomb & Viscous Friction	库仑和黏度摩擦非线性
	Dead Zone	死区非线性
up u lo y	Dead Zone Dynamic	动态死区非线性
	Hit Crossing	冲击非线性
	Quantizer	量化非线性
	Relay	继电非线性

续表

图　标	模 块 名	功　能
	Saturation	饱和非线性
up u lo y	Saturation Dynamic	动态饱和非线性

（5）离散系统模块

过程控制系统仿真中，离散系统模块（Discrete）中常用的子模块如表 5-5 所示。

表 5-5　常用的离散系统模块

图　标	模 块 名	功　能
$\frac{z-1}{z}$	Difference	差分环节
$\frac{K(z-1)}{Ts\,z}$	Discrete Derivative	离散微分环节
$\frac{1}{1+0.5z^{-1}}$	Discrete Filter	离散滤波器
$y(n)=Cx(n)+Du(n)$ $x(n+1)=Ax(n)+Bu(n)$	Discrete State-Space	离散状态空间系统模型
$\frac{1}{z+0.5}$	Discrete Transfer-Fcn	离散传递函数模型
$\frac{(z-1)}{z(z-0.5)}$	Discrete Zero-Pole	以零极点表示的离散传递函数模型
$\frac{K\,Ts}{z-1}$	Discrete-time Integrator	离散时间积分器
	First-Order Hold	一阶保持器
$\frac{0.05z}{z-0.95}$	Transfer Fcn First Order	离散一阶传递函数
$\frac{z-0.75}{z}$	Transfer Fcn Real Zero	离散零点传递函数
$\frac{1}{z}$	Unit Delay	一个采样周期的延迟
	Zero-Order Hold	零阶保持器

（6）输出显示部分模块

过程控制系统仿真中，输出显示部分模块常用的是接收器模块（Sinks），其中常用的子模块如表 5-6 所示。

表 5-6　常用的输出显示模块

图　　标	模 块 名	功　　能
	Display	数字显示器
	Floating Scope	浮动示波器
1	Out1	输出端口
	Scope	示波器
untitled.mat	To File(.mat)	将数据输出到文件中
simout	To Workspace	将数据输出到 MATLAB 的工作空间中
	XY Graph	在 MATLAB 图形窗口中显示信号的 X-Y 图

5.2 Simulink 仿真模型及仿真过程

1．Simulink 仿真模型组成

一个典型的 Simulink 仿真模型由以下三种类型的模块构成。

（1）信号源模块

信号源为系统的输入，它包括常数信号源、函数信号发生器（如正弦波和阶跃函数等）和用户自己在 MATLAB 中创建的自定义信号。

（2）被模拟的系统模块

系统模块作为仿真的中心模块，它是 Simulink 仿真建模所要解决的主要部分。

（3）输出显示模块

系统的输出由显示模块接收。输出显示的形式包括图形显示、示波器显示和输出到文件或 MATLAB 工作空间中三种，输出模块主要在 Sinks 库中。

构成 Simulink 仿真模型的三种模块的关联图如图 5-5 所示。

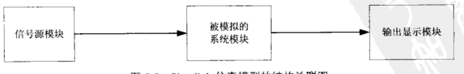


图 5-5　Simulink 仿真模型的结构关联图

Simulink 仿真模型的基本特点可归纳如下：

（1）Simulink 里提供了许多如 Scope（示波器）的接收器模块，这使得用 Simulink 进行仿真具有像做实验一般的图形化显示效果。

（2）Simulink 的模型具有层次性，通过底层子系统可以构建上层母系统。

（3）Simulink 提供了对子系统进行封装的功能，用户可以自定义子系统的图标和设置参数对话框。

2. Simulink 仿真的基本过程

启动 Simulink 后，便可在 Simulink 中进行建模仿真。Simulink 建模仿真的基本过程如下：

① 打开一个空白的 Simulink 模型窗口。

② 进入 Simulink 模块库浏览界面，将相应模块库中所需的模块拖到编辑窗口里。具体的操作是：用鼠标左键选中所需要的模块，然后将其拖到需要创建仿真模型的窗口，松开鼠标，这时所需要的模块就出现在 Simulink 模型窗口中。

③ 照给定的框图修改编辑窗口中模块的参数。在 Simulink 环境下绘制模块，只能绘出带有默认参数的模型，为了满足用户的具体需要，有时还需要对模块参数进行具体的设置。要对模块进行参数设置，首先双击该模块，打开此模块的参数设置对话框，然后在该参数设置对话框中，查看模块的各项默认参数设置，或者根据需要修改各项参数设置。

④ 将各个模块按给定的框图连接起来，搭建所需要的系统模型。

⑤ 用菜单或命令窗口输入命令进行仿真分析，在仿真的同时，可以观察仿真结果，如果发现有不正确的地方，可以停止仿真，对参数进行修正。

⑥ 如果对结果满意，可以将模型保存。

下面通过一个简单的模型来讲述 Simulink 建模仿真的基本操作过程。

【例 5-1】 利用 Simulink 设计一个简单的模型，其功能是将一个正弦信号输出到示波器中。

解： 解题的基本步骤如下所述。

① 新建一个模型窗口。

② 为模型添加所需模块。从源模块库（Sources）中复制正弦波模块（Sine Wave），从输出显示模块库（Sinks）复制示波器模块（Scope）。

③ 连接相关模块，构成所需要的系统模型，如图 5-6 所示。

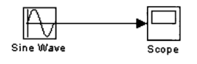


图 5-6 正弦信号输出到示波器中的模型

④ 进行系统仿真，单击模型窗口菜单中的 “Simulation” → “Start”，执行仿真。

⑤ 观察仿真结果。双击示波器模块，打开 Scope 窗口，结果为如图 5-7 所示的正弦波。

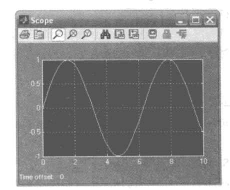


图 5-7　示波器中的仿真结果

5.3 Simulink 模块的处理

将仿真所需的模块从各自的模块库中拖到新建的模型窗口后，需要对这些模块进行处理，包括设置模块参数、连接模块等，然后才能构建仿真系统。

5.3.1 Simulink 模块参数设置

1．功能模块参数的设置

在设置功能模块参数后，才能进行仿真操作。不同功能模块的参数是不同的，用鼠标双击该功能模块会弹出相应的参数设置对话框。图 5-8 是传输延迟功能模块的对话框。

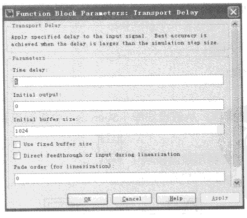


图 5-8　功能模块参数设置对话框

功能对话框由功能模块说明框和参数设置框组成。

（1）功能模块说明框用于说明该功能模块使用方法和功能。

（2）参数设置框用于设置该功能模块的参数。Simulink 中几乎所有模块的参数（Parameter）都允许用户进行设置，只要双击要设置参数的模块就会弹出设置对话框。

例如图 5-8 中，“Transport Delay”说明框中有该模块的英文说明，“Parameters”说明框由“Time Delay”（延迟时间）、“Initial output”（初始输出）、“Initial buffer size”（初始缓冲区的大小）和 Pade order（pade 近似的阶次）组成，用户可输入相关参数。每个对话框的下面有“OK”（确认）、“Cancel”（取消）、“Help”（帮助）和“Apply”（应用）4 个按钮。

- 设置功能模块参数后，需单击“OK”按钮进行确认，将设置参数送到仿真操作画面，并关闭对话框。
- 单击“Cancel”按钮将取消刚才输入的设置参数，并关闭对话框。
- 单击“Help”按钮，将弹出 Web 求助画面。
- 单击“Apply”按钮将设置参数送仿真操作画面，但不关闭参数设置对话框。

2. 示波器参数设置

采用 Simulink 仿真时，示波器可以接收向量信号，实时显示信号波形，但该波形不能直接打印或嵌入文件，示波器显示的结果直观且方便，是最常用的现实仿真结果的工具之一。

图 5-9 为示波器中显示的正弦波形，示波器窗口的标题是“Scope”，标题栏下是工具栏，下面简要介绍一下工具栏。

：这 3 个图标按钮分别管理 x-y 双向变焦（Zoom）、x 轴向变焦（Zoom X）、y 轴向变焦（Zoom y）。

：管理纵坐标的自动刻度（Autoscale），取当前信号的最大和最小值分别为纵坐标的上下限。

：把当前轴的设置保存为该示波器的默认设置。

：打开示波器属性对话框。

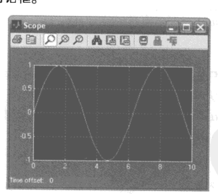


图 5-9 示波器中的正弦波形

在示波器“坐标框”内，单击鼠标右键，弹出一个现场菜单，选中菜单项“Axes properties”，又引出纵坐标设置对话框。在“Ymin”和“Ymax”栏中可填写所希望的纵轴下、上限。

单击示波器工具栏上的 按钮，打开如图 5-10 所示的示波器属性对话框。

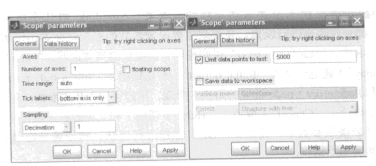


图 5-10　示波器属性对话框

影响横坐标显示的参数如下：

- Number of axes：多信号显示区设置。
- Time range：默认为 10，即显示在[0，10]区间的信号。
- Sampling：包含两个下拉菜单项，抽选（Decimation）和采样时间（Sample time）。"Decimation"设置显示频度，若取 n，表示每隔(n-1)个数据点给予显示；"Sampling time"设置显示点的采样时间步长。默认值为 0，表示显示连续信号。
- Limit row to last：设定缓冲区接收数据的长度。默认为勾选状态，其值为 5000。
- Save data to workspace：默认时，不被勾选。若该栏被勾选，则可把示波器缓冲区中保存的数据以矩阵或构架形式，送入 MATLAB 工作空间。

5.3.2　Simulink 模块基本操作

模块是建立 Simulink 模型的基本单元，用适当的方法把各种模块连接在一起就能够建立任何动态系统的模型。Simulink 模块的基本操作有选取、复制、删除、外形调整和模块名处理、颜色设定、属性设定等，下面分别进行介绍。

1．模块选取

当选取单个模块时，只要用鼠标在模块上单击即可，这时模块图标的 4 个角上出现黑色的小方块。选取多个模块时，在所有模块所占区域的一角按下鼠标左键不放，拖向该区域的对角，在此过程中会出现虚框，当虚框包住了要选的所有模块后，放开鼠标左键，这时在所有被选模块的图标上都会出现小黑方块，表示模块被选中了。

2．模块复制

模块可以在同一窗口内复制，也可以在不同窗口之间复制。

（1）在同一个模型窗口内复制

有时一个模型需要多个相同的模块，复制方法有如下 3 种：

- 用鼠标左键单击要复制的模块，按住左键，移动鼠标，同时按下 Ctrl 键，到适当位置释放鼠标，该模块就被复制到当前位置。
- 更简单的方法是，先选中模块，再按住鼠标右键（不按 Ctrl 键）移动鼠标，便可实

现复制。

- 还有一种方法是，先选定要复制的模块，选择“Edit”菜单下的“Copy”命令，然后选择“Paste”命令。

（2）在不同的窗口之间复制

当建立模型时，需要从模块库窗口，或者已经存在的窗口把需要的模块复制到新建模型文件的窗口。要对已经存在的模块进行编辑时，有时也需要从模块库窗口或另一个已经存在的模型窗口复制模块。复制方法有如下 2 种：

- 最简单的办法是，用鼠标左键选择要复制的模块（首先要打开源模块和目标模块所在的窗口），按住左键，移动鼠标到相应窗口（不用按住 Ctrl 键），然后释放，该模块就会被复制过来，而源模块不会被删除。
- 使用“Edit”菜单的“Copy”和“Paste”命令来完成复制。先选定要复制的模块，选择“Edit”菜单下的“Copy”命令，然后切换到目标窗口的“Edit”菜单下选择“Paste”命令。



复制结果中会发现复制出的模块名称在原名称的基础上又加了编号，这是 Simulink 的约定，每个模型中的模块和名称是一一对应的，相同的模块或不同的模块都不能用同一个名字。

3. 模块删除

选中模块，按 Delete 键即可。若要删除多个模块，可以同时按住 Shift 键，再用鼠标选中多个模块，按 Delete 键即可；也可以用鼠标选取某区域，再按 Delete 键就可以把该区域中的所有模块和线等全部删除。

4. 模块外形的调整

模块外形的调整有大小的改变、方向的改变和添加阴影。

（1）改变大小

选定模块，用鼠标选择其周围的四个黑方块中的任意一个，拖动鼠标，这时会出现虚线的矩形表示新模块的位置，调整到需要的大小后释放鼠标即可。

（2）改变方向

为了能够顺序连接功能模块的输入和输出端，功能模块有时需要转向。在菜单“Format”中选择“Flip Block”旋转 180°，选择“Rotate Block”顺时针旋转 90°；或者直接按 Ctrl+F 组合键执行“Flip Block”，按 Ctrl+R 组合键执行“Rotate Block”。

（3）给模块加阴影

选定模块，选取菜单“Format”下的“Show Drop Shadow”使模块产生阴影效果。

5. 模块名的处理

模块名的处理包括名称的显示与否、修改和改变模块名位置等。

（1）是否显示模块名

选取菜单“Format”下的“Hide Name”，模块名就会被隐藏，同时“Hide Name”改为“Show Name”。

选取“Show Name”就会使模块隐藏的名字显示出来。

（2）修改模块名

单击模块名的区域，会在此处出现编辑状态的光标，在这种状态下能够对模块名随意修改。

模块名和模块图标中的字体也可以更改，方法是选定模块，在菜单“Format”下选取“Font”，这时会弹出“Set Font”对话框，在对话框中可选取想要的字体。

（3）改变模块名的位置

模块名的位置有一定的规律，当模块的接口在左右两侧时，模块名只能位于模块的上下两侧，默认在下侧；当模块的接口在上下两侧时，模块名只能位于模块的左右两侧，默认在左侧。

因此模块名只能从原位置移到相对的位置。可以用鼠标拖动模块名到其相对的位置；也可以选定模块，用菜单“Format”下的“Flip Name”命令实现相同的移动。

6. 模块颜色的设置

“Format”菜单中的“Foreground Color”命令可以改变模块的前景颜色，“Background Color”命令可以改变模块的背景颜色，而模型窗口的颜色可以通过“Screen Color”命令来改变。

7. 模块属性的设置

选中模块，通过“Edit”菜单的“Block Properties”命令可以对模块进行属性的设置，包括对 Description、Priority、Tag、Open function、Attributes format string 等属性的设置。

其中 Open function 属性是一个很有用的属性，通过它指定一个函数名；当模块被双击之后，Simulink 就会调用该函数并执行，这种函数在 MATLAB 中称为回调函数。

5.3.3 Simulink 模块连接

上面介绍了对模块本身的各种操作，当设置好各个模块后，还需要把它们按照一定的顺序连接起来才能组成一个完整的系统模型。

1. 模块间连线

在模块间连线，有以下几种情况。

（1）连接两个模块

从一个模块的输出端连到另一个模块的输入端，这是最基本的连接情况。方法是将光标移动到输出端，光标的箭头会变成十字形光标，这时按住鼠标左键不放，移动光标到另一个模块的输入端，当十字光标出现“重影”时，释放鼠标左键就完成了连接。

如果两个模块不在同一水平线上，连线是一条折线。如果要用斜线表示，必须在连接时按住 Shift 键。

（2）模块间连线的调整

调整模块间连线位置可利用鼠标简单的拖动来实现，即先把光标移到需要移动的线段的位置，按住鼠标左键，再把光标移动到目标位置，释放鼠标左键。

还有一种情况是要把一条直线分成斜线段，调整方法和前一种情况类似，不同之处在于按住鼠标之前要先按下 Shift 键，出现小黑方框之后，用鼠标左键选择小黑方框，按下左键不放并移动鼠标，移动到适当的位置后释放 Shift 键和鼠标。

（3）在连线之间插入模块

把该模块用鼠标拖到连线上，然后释放鼠标即可。

（4）连线的分支

这是经常会碰到的一些情况，即需要把一个信号输送到不同的模块，这时就需要使用分支结构的连线。例如要把正弦波信号实时显示出来，同时还要将信号数据存到文件。

操作步骤是：在先连好一条线以后，把鼠标移到支线的起点位置，先按下鼠标左键，然后按住 Ctrl，将鼠标拖到目标模块的输入端，最后释放鼠标和 Ctrl 键。

2. 在连线上标示信息

在连线上标示信息包括标示向量、显示数据类型和信号标记等。

（1）标示向量

为了能比较直观地区别各个模块之间传输的数据是数据还是矩阵（向量），可以选择模型文件菜单“Format”下的“Wide vector Lines”选项，这样传输向量的连线就会变粗。

如果再选择“Format”下的“Vector Lines Widths”选项，在传输矩阵的连线上方会显示出通过该连线的矩阵维数。

（2）显示数据类型

在连线上可以显示一个模块输出的数据类型，选择菜单“Format”下的“Port Data Types”选项即可实现。

（3）信号标记

为了使模型更加直观、可读性更强，可以为传输的信号做标记。建立信号标记的办法是：双击要做标记的线段，出现一个小文本编辑框，在里面输入标记的文本，这样就建立了一个信号标记。信号标记可以随信号的传输在一些模块中传递。

支持这种传递的模块有 Mux、Demux、Inport、From、Selector、Subsystem 和 Enable。

要实现信号标记的传递，需要在上面列出的某个模块的输出端建立一个以“<”开头的标记。当开始仿真或执行“Edit”菜单下的“Updata Diagram”命令时，传输过来的信号标记就会显示出来。

5.4 Simulink 仿真设置

在编辑好仿真程序后，应设置仿真操作参数，以便进行仿真。单击"Simulation"菜单下面的"Configuration Parameters"项或者直接按快捷键 Ctrl+E，便弹出如图 5-11 所示的设置界面，它包括仿真器参数(Solver)设置、工作空间数据导入/导出(Data Import/Export)设置等。下面对控制系统仿真中常用的仿真设置进行介绍。

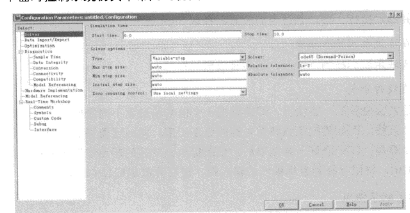


图 5-11 Simulink 设置窗口

5.4.1 仿真器参数设置

可以设置仿真开始时间、仿真结束时间、解法器及输出项等仿真器参数。对于一般的仿真，使用默认设置即可。

1. 仿真时间（Simulation time）设置

这里所指的时间概念与真实的时间并不一样，只是计算机仿真中对时间的一种表示，比如 10s 的仿真时间，如果采样步长定为 0.1，则需要执行 100 步，若把步长减小，则采样点数增加，那么实际的执行时间就会增加。

通常需要设置仿真开始时间 (Start time) 和仿真结束时间 (Stop time)。一般仿真开始时间设为 0，而结束时间则视不同的情况进行选择。

总的说来，执行一次仿真要耗费的时间取决于很多因素，包括模型的复杂程度、解法器及其步长的选择、计算机时钟的速度等。

2. 仿真步长模式设置

在图 5-11 所示界面"Type"下拉选项框中指定仿真的步长方式，可供选择的有 Variable-step (变步长) 和 Fixed-step (固定步长) 方式。

选择变步长模式则可以在仿真过程中改变步长，提供误差控制和过零检测选择。固定步长模式则可以在仿真过程中提供固定的步长，不提供误差控制和过零检测。

3. 解法器设置

在“Solver”下拉选项框中选择解法器。可选的变步长模式解法器有：discrete、ode45、ode23、ode113、ode15s、ode23s、ode23t 和 ode23tb。下面简要概述一下这些解法器的含义。

（1）discrete：当 Simulink 检查到模型没有连续状态时使用它。

（2）ode45：默认值，表示四/五阶龙格-库塔法，适用于大多数连续或离散系统，但不适用于刚性（stiff）系统。它是单步解法器，即在计算 $y(t_n)$ 时，它仅需要最近处理时刻的结果 $y(t_{n-1})$。一般来说，面对一个仿真问题最好首先试试 ode45。

（3）ode23：表示二/三阶龙格-库塔法，它在误差限要求不高和求解的问题不太难的情况下，可能会比 ode45 更有效。它也是一个单步解法器。

（4）ode113：表示一种阶数可变的解法器，它在误差容许要求严格的情况下通常比 ode45 有效。ode113 是一种多步解法器，即在计算当前时刻输出时，它需要以前多个时刻的解。

（5）ode15s：表示一种基于数字微分公式的解法器（NDFs），它也是一种多步解法器，适用于刚性系统。当用户估计要解决的问题是比较困难的、不能使用 ode45 或者即使使用效果也不好时，就可以用 ode15s。

（6）ode23s：表示一种单步解法器，专门应用于刚性系统，在弱误差允许下的效果优于 ode15s。它能解决某些 ode15s 所不能有效解决的刚性问题。

（7）ode23t：表示梯形规则的一种自由插值实现。这种解法器适用于求解适度刚性而用户又需要一个无数字振荡的解法器的情况。

（8）ode23tb：表示 TR-BDF2 的一种实现，TR-BDF2 是具有两个阶段的隐式龙格-库塔公式。

固定步长模式解法器有：discrete、ode5、ode4、ode3、ode2、ode1 和 ode14x。

（1）discrete：表示一种实现积分的固定步长解法器，它适合于离散无连续状态的系统。

（2）ode5：默认值，是 ode45 的固定步长版本，适用于大多数连续或离散系统，不适用于刚性系统。

（3）ode4：表示四阶龙格-库塔法，具有一定的计算精度。

（4）ode3：表示固定步长的二/三阶龙格-库塔法。

（5）ode2：表示改进的欧拉法。

（6）ode1：表示欧拉法。

（7）ode14x：表示固定步长的隐式外推法。

4. 变步长的参数设置

对于变步长模式，用户常用的设置有：“Max step size”（最大步长参数）和“Min step size”（最小步长参数）、“Relative tolerance”（相对误差）和“Absolute tolerance”（绝对误差）、初始步长，以及“Zero crossing control”（过零控制）。默认情况下，步长自动确定，用 auto 值表示。

（1）Max step size：决定解法器能够使用的最大时间步长，它的默认值为“仿真时

间/50"，即整个仿真过程中至少取 50 个取样点。这样的取法对于仿真时间较长的系统则可能带来取样点过于稀疏的问题，继而使仿真结果失真。一般建议对于仿真时间不超过 15s 的采用默认值即可，对于超过 15s 的每秒至少保证 5 个采样点，对于超过 100s 的，每秒至少保证 3 个采样点。

（2）Min step size：用来规定变步长仿真时使用的最小步长。

（3）Relative tolerance：指误差相对于状态的值，是一个百分比，默认值为 1e–3，表示状态的计算值要精确到 0.1%。

（4）Absolute tolerance：表示误差值的门限，或者是在状态值为零的情况下可以接受的误差。如果它被设成了 auto，那么 Simulink 为每一个状态设置初始绝对误差为 1e–6。

（5）Initial step size：一般建议使用 auto 默认值。

（6）Zero crossing control：过零点控制，用来检查仿真系统的非连续。

5. 固定步长的参数设置

对于固定步长模式，用户常用的设置如下所述。

（1）Multitasking：选择这种模式时，当 Simulink 检测到模块间非法的采样速率转换时会给出错误提示。所谓的非法采样速率转换，指两个工作在不同采样速率的模块之间的直接连接。在实时多任务系统中，如果任务之间存在非法采样速率转换，那么就有可能出现一个模块的输出在另一个模块需要时却无法利用的情况。

通过检查这种转换，Multitasking 将有助于用户建立一个符合现实的多任务系统的有效模型。使用速率转换模块可以减少模型中的非法速率转换。Simulink 提供了两个这样的模块：unit delay 模块和 zero-order hold 模块。对于从慢速率到快速率的非法转换，可以在慢输出端口和快输入端口插入一个单位延时（unit delay）模块。对于快速率到慢速率的转换，则可以插入一个零阶采样保持器（zero-order hold）。

（2）Singletasking：这种模式不检查模块间的速率转换，它在建立单任务系统模型时非常有用，在这种系统中不存在任务同步问题。

（3）Auto：选择这种模式时，Simulink 会根据模型中模块的采样速率是否一致，自动决定切换到 Multitasking 模式或 Singletasking 模式。

5.4.2 工作空间数据导入/导出设置

工作空间数据导入/导出（Data Import/Export）设置主要在 Simulink 与 MATLAB 工作空间交换数值时进行有关选项设置，可以设置 Simulink 和当前工作空间的数据输入或输出。

通过设置，可以从工作空间输入数据、初始化状态模块，也可以把仿真结果、状态变量、时间数据保存到当前工作空间，它包括 Load from workspace、Save to workspace 和 Save option 三个选择项。

（1）Load from workspace：勾选该选项，即可从 MATLAB 工作空间获取时间和输入变量，一般时间变量定义为 *t*，输入变量定义为 *u*。"Initial state" 用来定义从 MATLAB 工作空间获得状态初始值的变量名。

Simulink 通过设置模型的输入端口，实现在仿真过程中从工作空间读入数据，常用的输入端口模块为信号与系统模块库（Signals & Systems）中的“In1”模块，设置其参数时，勾选“input”前的复选框，并在后面的编辑框输入数据的变量名，并可以用命令窗口或 M 文件编辑器输入数据。Simulink 根据输入端口参数中设置的采样时间读取输入数据。

（2）Save to workspace：用来设置存在 MATLAB 工作空间的变量类型和变量名，可以选择保存的选项有时间、端口输出、状态和最终状态。勾选该选项前面的复选框，并在选项后面的编辑框输入变量名，就会把相应数据保存到指定的变量中。常用的输出模块为信号与系统模块库中的“Out1”模块和输出方式库（Sink）中的“To Workspace”模块。

（3）Save options：用来设置存往工作空间的有关选项。

- Limit rows to last 用来设置 Simulink 仿真结果最终可存往 MATLAB 工作空间的变量的规模，对于向量而言即其维数，对于矩阵而言即其秩。
- Decimation 设定了一个亚采样因子，它的默认值为 1，也就是对每一个仿真时间点产生值都保存；若该值为 2，则每隔一个仿真时刻才保存一个值。
- Format 用来说明返回数据的格式，包括数组（Array）、结构体（Structure）及带时间的结构体（Structure with time）。
- Signal logging name 用来保存仿真中记录的变量名。
- Output options 用来生成额外的输出信号数据，它有以下 3 个选项。
  - ➢ Refine output：这个选项可以理解成精细输出，其意义是在仿真输出太稀松时，Simulink 会产生额外的精细输出，这一点就像插值处理一样。用户可以在“Refine factor”设置仿真时间步间插入的输出点数，要产生更光滑的输出曲线，改变精细因子比减小仿真步长更有效。精细输出只能在变步长模式中才能使用，并且在 ode45 效果最好。
  - ➢ Produce additional output：它允许用户直接指定产生输出的时间点。一旦选择了该项，则在它的右边出现一个“output times”编辑框，在这里用户指定额外的仿真输出点，它既可以是一个时间向量，也可以是表达式。与精细因子相比，这个选项会改变仿真的步长。
  - ➢ Produce specified output only：它的意思是让 Simulink 只在指定的时间点上产生输出。为此，解法器要调整仿真步长，使之和指定的时间点重合。这个选项在比较不同的仿真时可以确保它们在相同的时间输出。
- Refine factor 用来指定仿真步长之间产生数据的点数。

5.5 Simulink 仿真举例

通过前面的内容，你应该了解并初步掌握了 Simulink 的使用。使用 Simulink 仿真的基本步骤如下：

① 启动 Simulink 并打开模型编辑窗口。

② 将所需模块添加到模型中。

③ 设置模块参数并连接各个模块组成仿真模型。

④ 设置系统仿真参数。

⑤ 开始系统仿真。

⑥ 观察仿真结果。

下面通过几个实例，讲述如何使用 Simulink 进行仿真。

【例 5-2】 使用 Simulink 产生一个 2s 时出现的单位阶跃输入信号，并在示波器中显示出来。

解：本题的基本求解步骤如下：

① 利用 Simulink 的“Library”窗口中的“File”→“New”，打开一个新的模型窗口。

② 分别从信号源库（Sources）、输出方式库（Sinks）中，用鼠标把阶跃信号发生器（Step）、示波器（Scope）这两个标准功能模块选中，并将其拖至模型窗口。

③ 按要求直接将 Step 与 Scope 相连。

④ 双击 Step 模块，打开其属性设置对话框，并将其中的“Step time”设置为“2”，其他参数为默认值，如图 5-12 所示。

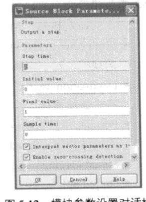


图 5-12　模块参数设置对话框

⑤ 绘制成功后，如图 5-13 所示，为此文件命名并保存。

⑥ 单击 ▶ 按钮对模型进行仿真，运行后，双击示波器，得到图形如图 5-14 所示。

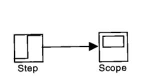


图 5-13　显示单位阶跃信号的 Simulink 模型

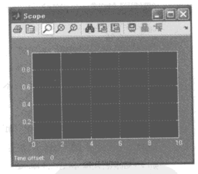


图 5-14　示波器输出结果

从示波器中可以看出，该信号是 2s 的时产生的幅值为 1 的阶跃信号。

【例 5-3】 使用 Simulink 求解微分方程：$\frac{\mathrm{d}u}{\mathrm{d}t}=\cos(\sin t)$，$u(0)=1$。

解：从数学的角度看，要由 t 得到 u 的数值解，需要先对 $\sin t$ 进行余弦运算，然后再积分。在弄清数学模型结构之后，就可以根据数学模型设计相应的仿真模型。

在本例中，需要正弦信号、余弦函数、积分模块、观测结果的模块。从“Simulink Library Browser”中分别将这些模块依次拖到“untitled”窗口中，如图 5-15 所示。其中有：

- Sources 模块组中的 Sine Wave 模块，它产生正弦信号。
- Math Operations 模块组中的 Trigonometric Function 模块，这是一个三角函数模块，双击该模块，在弹出的对话框中选择余弦函数（cos）。
- Continuous 模块组中的 Integrator 模块，它是积分模块，由于函数 u 的初始值为 1，也就是说，积分器的初始条件为 1，因此，在 Integrator 的属性设置中将“Initial condition”设置为 1。
- Sinks 模块组中的 Scope 模块，这是示波器模块，能将输出结果显示出来。

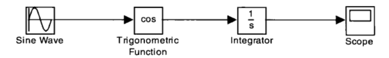


图 5-15 求解微分方程的 Simulink 模型

按照题意连接好模块后，在默认参数下运行，输出结果如图 5-16 所示。

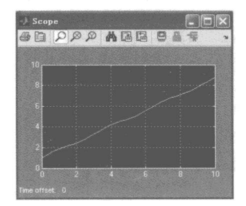


图 5-16 微分方程求解的结果

【例 5-4】 产生一个 5sin(2t)和 sin(5t)叠加的信号，而且还叠加功率谱为 0.5 的限带宽白噪声。

解：首先需要产生 5sin(2t)、sin(5t)和限带宽白噪声信号，然后将这 3 个信号叠加起来。在本例中，需要正弦信号、限带宽白噪声、加法模块、观测结果的模块。从“Simulink Library Browser”中分别将这些模块依次拖到“untitled”窗口中，如图 5-17 所示。其中：

- Sources 模块组中的 Signal Generator 模块，它是信号发生器模块。双击该模块，在弹出的属性设置框中，选择 sine 波形，设置幅值为 5，频率为 2，使之产生 5sin(2t)信号；对另一个信号发生器模块做类似的设置，使之产生 sin(5t)信号。
- Sources 模块组中的 Band-Limited White Noise 模块，它是限带宽白噪声模块。双击该模块，在弹出的属性设置框中设置“Noise power”（功率谱）为 0.5，产生功率谱为 0.5 的限带宽白噪声信号。
- Math Operations 模块组中的 Add 模块，它是加法模块，默认是两个输入相加。双击

该模块，在弹出的属性设置框中将“List of Signs”框中的两个加号（++）改为三个加号（+++），表明对 3 个输入量进行相加。

- Sinks 模块组中的 Scope 模块，这是示波器模块，能将输出结果显示出来。

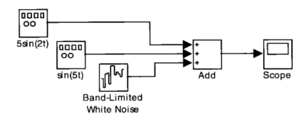


图 5-17　信号叠加的 Simulink 模型

按照题意连接好模块后，在默认参数下运行，输出结果如图 5-18 所示。

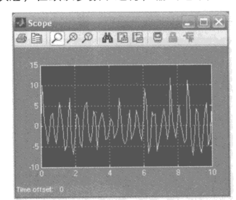


图 5-18　信号叠加的结果

通过以上几个简单例子的讲述，可以总结出利用 Simulink 进行系统仿真的 3 大步骤：

① 建立系统仿真模型，包括添加模块、设置模块参数以及进行模块连接等操作。

② 设置仿真参数。

③ 启动仿真并分析仿真结果。

5.6 小结

Simulink 是一个非常友好的仿真工具，入门简单，使用方便。了解 Simulink 的构成，掌握 Simulink 的基本操作，是使用 Simulink 进行系统仿真的基础。

出于篇幅的考虑，本章只介绍了 Simulink 最常用、最具有代表性内容，对于其他内容，读者可在掌握这些基本内容的基础上，通过 Help 文档和反复练习，触类旁通，达到熟练掌握、运用自如的境界。

Part 2

神经网络提高篇

第 6 章　MATLAB 神经网络工具箱概述

神经网络工具箱是 MATLAB 功能强大、应用广泛的工具箱之一，应用 MATLAB 神经网络工具箱，可以方便地构造出适用于特定问题的神经网络，这为神经网络的实现提供了一个非常便利的操作环境和手段，对神经网络的应用有很大的帮助。

6.1 神经网络简介

人工神经网络（Artificial Neural Network，ANN），亦称为神经网络（Neural Network，NN），是一种应用类似于大脑神经突触连接的结构进行信息处理的数学模型，在工程与学术界也常直接简称为“神经网络”或“类神经网络”。它是对人脑的抽象、简化和模拟，反映了人脑的基本特性。神经网络的研究是从人脑的生理结构出发来研究人的智能行为，模拟人脑信息处理的功能。它是根植于神经科学、数学、统计学、物理学、计算机科学，以及工程学科的一门技术。

神经网络是一种运算模型，由大量的节点（或称神经元或单元）和相互之间的加权连接构成。每个节点代表一种特定的输出函数，称为激励函数（activation function）。每两个节点间的连接都代表一个对于通过该连接信号的加权值，称之为权重（weight），这相当于神经网络的记忆。网络的输出则根据网络的连接方式、权重值和激励函数的不同而不同。而网络自身通常都是对自然界某种算法或者函数的逼近，也可能是对一种逻辑策略的表达。

它的构筑理念是受到生物（人或其他动物）的神经网络功能的运作启发而产生的。人工神经网络通常是通过一个基于数学统计学类型的学习方法（Learning Method）得以优化，所以神经网络也是数学统计学方法的一种实际应用，通过统计学的标准数学方法我们能够得到大量的可以用函数来表达的局部结构空间；另一方面在人工智能学的人工感知领域，我们通过数学统计学的应用可以来做人工感知方面的决定问题（也就是说通过统计学的方法，神经网络能够具备类似人一样具有简单的决定能力和简单的判断能力），这种方法比起正式的逻辑学推理演算更具有优势。

神经网络也常常被称做神经计算机，但它与现代数字计算机是有所不同的，主要体现在以下方面：

（1）非线性。非线性关系是自然界的普遍特性。大脑的智慧就是一种非线性现象。人工神经元处于激活或抑制两种不同的状态，这种行为在数学上表现为一种非线性关系。具有阈值的神经元构成的网络具有更好的性能，可以提高容错性和存储容量。

（2）非局限性。一个神经网络通常由多个神经元广泛连接而成。一个系统的整体行为不仅取决于单个神经元的特征，而且可能主要由单元之间的相互作用、相互连接所决定。

通过单元之间的大量连接，模拟大脑的非局限性。联想记忆是非局限性的典型例子。

（3）非常定性。神经网络具有自适应、自组织、自学习能力。不仅神经网络处理的信息可以有各种变化，而且在处理信息的同时，非线性动力系统本身也在不断变化。通常采用迭代过程描写动力系统的演化过程。

（4）非凸性。一个系统的演化方向，在一定条件下将取决于某个特定的状态函数。例如能量函数，它的极值就对应于系统比较稳定的状态。非凸性是指这种函数有多个极值，故系统具有多个较稳定的平衡态，这将导致系统演化的多样性。

因为借鉴了生物脑神经的研究成果，神经网络成为 20 世纪 80 年代再度活跃起来的新的信息处理科学研究领域。这与神经网络是以非线性处理为基础分不开的，正是由于非线性作用，才形成了大到大自然，小到工程技术等系统的复杂性。

神经网络涉及诸多学科的知识以及应用领域，并在不断的扩展之中。本章将从发展历史、模型、分类、应用，以及 MATLAB 工具箱的角度对神经网络进行讲述，以便读者在开始工具箱实例的学习之前对神经网络理论有一个整体的把握。

6.2 神经网络模型及训练

6.2.1 生物神经元模型

人脑是自然界所造就的最高级产物。人的思维是由人脑来完成的，而思维是人类智能的集中体现。人的思维可概括为逻辑思维和形象思维两种，前者主要由左脑掌管，后者则由右脑负责。

医学研究表明，人类大脑皮层中大约包含 100 亿个神经元、60 万亿个神经突触，以及它们的连接体。神经元之间通过相互连接形成错综复杂而又灵活多变的神经网络系统。

神经系统的基本结构和功能单位是神经细胞，即神经元（neurons），它主要由细胞体、树突、轴突和突触组成，如图 6-1 所示。

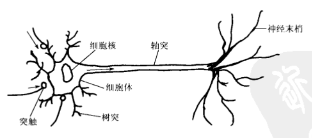


图 6-1 生物神经元模型

细胞体除细胞核外，还有线粒体、高尔基体、尼氏体（Nissl's bodies）等。尼氏体颗粒状，是糙面内质网和游离核糖体的混合物，神经元的各种蛋白质都是在这里合成的。细胞质中还有不同走向的微管、微丝和密布的中间纤维，即神经元纤维（neurofilaments）。

它们构成神经元的骨架，有保持神经元形态的作用。微管还有运输物质的功能。

神经元伸出的突起分为两种，即树突和轴突。树突（dendrites）短而多分支，每支可再分支，尼氏体可深入树突中，树突和细胞体的表膜都有接受刺激的功能。它们的表面富有小棘状突起，是与其他神经元的轴突相连（突触）之处。轴突和树突在形态和功能上都不相同。每一个神经元一般只有一个轴突，从细胞体的一个凸出部分伸出。轴突不含尼氏体，轴突表面也无棘状突起。轴突一般都比树突长，其功能是把从树突和细胞表面传入细胞体的神经冲动传到其他神经元或效应器。所以，树突是传入纤维，轴突是传出纤维。

有些神经元有一个轴突和一个树突，称为两极神经元；有些神经元有一个轴突和多个树突，称为多极神经元；还有些神经元只有一条纤维，称为单极神经元。如人和其他脊椎动物脊神经中的感觉神经元只伸出一条纤维，但在离开细胞体不远处分为两支。一支到感受器，称为周围支，另一支进入脊髓，称为中枢支。前者是传入纤维，将来自感受器的感觉冲动传入细胞体，所以从功能上看应是树突，但却有轴突的结构；后者从功能和结构上看都肯定是轴突。

神经元具有以下基本功能特性：

- 时空整合功能；
- 神经元的动态极化性；
- 兴奋与抑制状态；
- 结构的可塑性；
- 脉冲与电位信号的转换；
- 突触延期和不延期；
- 学习、遗忘和疲劳。

神经网络是由大量的神经元单元互相连接而构成的网络系统。但是，实际上神经网络的模型只是生物神经网络的抽象、简化和模拟，并不能够完全反映大脑的功能，而神经网络模型的理念正是受到生物神经网络模型的研究启发而产生的。

6.2.2 神经网络模型

人工神经网络，是通过模仿动物神经网络行为特征，进行分布式并行信息处理的算法数学模型。这种网络依靠系统的复杂程度，通过调整内部大量节点之间相互连接的关系，从而达到处理信息的目的。

人工神经网络具有自学习和自适应的能力，可以通过预先提供的一批相互对应的输入-输出数据，分析掌握两者之间潜在的规律，最终根据这些规律，用新的输入数据来推算输出结果，这种学习分析的过程被称为"训练"。

同生物神经系统类似，人工神经网络也是由人工神经元为基本单元构成的。人工神经元是模拟生物神经元的数学模型，是人工神经网络的基本处理单元，同时也是一个多输入/单输出的非线性元件，如图 6-2 所示。

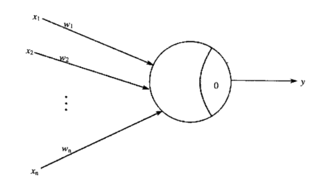


图 6-2 简单人工神经元模型

神经元的每一个输入连接都有突触连接强度，用一个连接权值来表示，即将产生的信号通过连接强度放大，每一个输入量（x_j）都相应有一个相关联的权重（w_{ij}）。处理单元将经过权重的输入量化，然后相加求得其加权值之和，计算出唯一的输出量，这个输出量（y）是权重和的函数，一般称此函数为传递函数。这个过程可以用公式表示为：

$$y = f(\sum_j w_{ij} x_j + b)$$

其中 f 表示此神经元采用的传递函数。

对于基本的感知器神经元，其工作方式是将加权总和与神经元的阈值进行比较，若它大于阈值，神经元被激活。当它被激活时，信号被传送到与其相连的更高一级神经元。此过程中采用的传递函数为硬限值函数。

不同类型的神经网络采用的传递函数各有不同，通常采用的函数包括：硬限值函数（hardlim）、线性函数（purelin）、Sigmoid 函数（logsig）、高斯径向基函数，等等。传递函数的不同也导致了各种神经网络在结构和功能上的差异，如图 6-3 所示。

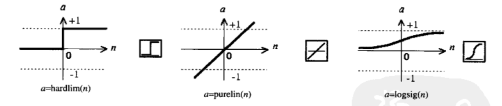


图 6-3 常用的传递函数示意

神经网络由排列成层的处理单元组成，接收输入信号的神经元层称输入层，输出信号的神经元层称输出层，不直接与输入/输出发生联系的神经元层称为中间层或隐层。

如果网络获得一组输入数据，在网络输入层的每个神经元都接收到输入模式的一部分，然后输入层将输入通过连接传递给隐层。隐层接收到整个输入模式，由于传递函数的作用，隐层单元的输出就与输入层大不相同。

输出单元从隐层单元接收输出活动的全部模式，但隐层单元往输出层的信号传递要经

过权重的连接，所以输出层单元有的激发，有的抑制，从而产生相应的输出信号。输出层单元输出的模式就是网络对输入模式激励的总的响应。

基本的神经网络结构如图 6-4 所示。

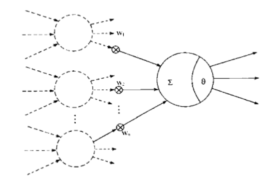


图 6-4 神经网络结构示意

对于大多数神经网络，在网络运行的时候，传递函数一旦选定，就保持不变。而权重的动态修改是学习中最基本的过程，类似于“智能过程”。可见，网络最重要的信息是存储于调整过的权重之中。

6.2.3 神经网络的训练

当神经网络的模型结构确定之后，接下来就是学习和训练。网络不是通过修改处理单元本身来完成训练过程，而是靠改变网络中连接的权重来学习的。因此若处理单元要学会正确地反映所给数据的模式，唯一用以改变处理单元性能的元素就是连接的权重。

应该指出，训练和学习并不完全相同，训练是指神经网络的学习过程，而学习是此过程的结果。训练是外部过程，而学习是神经网络的内部过程。

由于神经网络的连接模型、输入信息的离散性或连续性、有无监督训练、神经元的动态特性等的不同，响应的学习算法也不同。

（1）有监督学习算法。要求同时给出输入和正确的输出，网络根据当前输出与所要求的目标输出差来进行网络调整，使网络做出正确的反应。

（2）无监督学习算法。只需给出一组输入，网络能够逐渐演变对输入的某种模式做出特定的反应。即训练样本中只有输入矢量，神经网络靠比较各输入矢量之间的关系来调整权值，从而将训练样本中输入矢量之间的关系映射到网络的权值上，以实现联想记忆或数据压缩的功能。例如人脑采用的就是一种典型的无监督训练。由于这类网络没有输出矢量，所以不能用于模拟函数。

神经网络的学习规则对应着各种不同的调整神经元之间权重的策略，包括如下方面：

- Hebb 学习规则。
- 纠错学习规则，也称 Widrow-Hoff 学习规则。
- 基于记忆的学习规则。

- 随机学习规则。
- 竞争学习规则。

6.2.4 神经网络的分类

神经网络通常按照不同的结构、功能，以及学习算法，对网络进行分类，可以分为：

1．感知器神经网络

最简单的神经网络类型，只有单层的神经网络结构，采用硬限值函数作为网络传递函数，适用于简单的线性二类划分问题。

2．线性神经网络

单层结构的神经网络，采用线性函数作为网络传递函数，适用于数据的线性拟合与逼近等应用。

3．BP 神经网络

应用最为广泛的网络，具有多层的网络结构，含有一个或更多的隐层。其结构如图 6-5 所示。

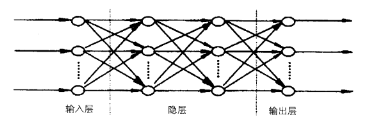


图 6-5 BP 网络结构

BP 网络采用 Widrow-Hoff 学习算法和非线性传递函数，典型的 BP 网络采用的是梯度下降算法，也就是 Widrow-Hoff 算法所规定的。BP，即 Back Propagation，就是指为非线性多层网络训练中梯度计算时采用信号正向传播、误差反向传播的方式。

对于 BP 网络的训练，现在有许多基本的优化算法，例如变尺度算法和牛顿算法。MATLAB 神经网络工具箱提供了许多这样的算法。

通过采用非线性传递函数，BP 网络能够以任意的精度逼近任何非线性函数，这是一项非常重要的性能。由于采用隐层中间层结构，BP 网络能够提取出更高阶的统计性质，尤其是当输入规模庞大时，网络能够提取高阶统计性质的能力就显得非常重要。

4．径向基神经网络

径向基神经网络又称为 RBF 网络，它与 BP 网络同为多层前向网络，也能够以任意的精度逼近任何非线性函数。只是两者采用的传递函数不同。BP 网络通常采用 Sigmoid 函数或线性函数作为传递函数，而 RBF 网络则采用径向基函数作为传递函数。

由于采用了径向基传递函数，RBF 网络具有比 BP 网络更高的训练速度，避免了误差反向传播过程中烦琐的计算，更适用于新数据，收敛性也相对较好。

5. 竞争神经网络

竞争神经网络的特点是它的各个神经元之间是相互竞争的关系，众多神经元之间相互竞争以决定出胜者，获胜神经元决定哪一种原型模式最能够代表输入模式。因此竞争神经网络非常适应于模式分类的问题。

常见的竞争神经网络包括自组织映射网络、学习矢量量化网络，等等。

6. 反馈神经网络（Hopfield 神经网络）

反馈神经网络是一种反馈动力学系统。在这种网络中，每个神经元同时将自身的输出信号作为输入信号反馈给其他神经元，它需要工作一段时间才能达到稳定。Hopfield 神经网络是反馈网络中最简单且应用广泛的模型，它具有联想记忆的功能，如果将李雅普诺夫函数定义为寻优函数，Hopfield 神经网络还可以用来解决快速寻优问题。其结构如图 6-6 所示。

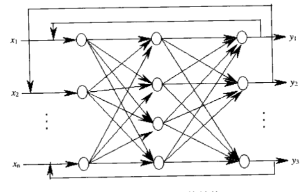


图 6-6　反馈神经网络结构

7. 随机神经网络

随机神经网络是在神经网络中引入了随机机制而产生的，一个经典的范例是采用基于统计物理的模拟退火算法进行训练的 Boltzmann 随机网络模型。在随机神经网络中，神经元按照概率原理工作，网络也依照概率取得不同的网络状态。

随机神经网络常用来解决组合优化的问题。

6.3 神经网络的应用

神经网络的发展已经有 50 年的历史了，但是实际的应用却是在最近 15 年，如今神经网络仍快速发展着，其应用的种类已经涵盖到各行各业方方面面，下面列举几个神经网络的应用方向对其进行说明。

1. 工业控制应用

基于人工神经网络的控制（ANN-based Control）简称神经控制（Neural Control）。利用神经网络强大的自适应性和学习能力、非线性映射能力、鲁棒性和容错能力，充分地将这些神经网络特性应用于控制领域，可使控制系统的智能化向前迈进一大步。

随着被控系统越来越复杂，人们对控制系统的要求越来越高，特别是要求控制系统能适应不确定性、时变的对象与环境。传统的基于精确模型的控制方法难以适应要求，现在关于控制的概念也已更加广泛，它要求包括一些决策、规划以及学习功能。神经网络由于具有上述优点而越来越受到人们的重视。

2. 医学应用

随着神经网络新的理论、技术、方法的不断涌现，其模仿人的智能的程度不断提升，已经得到很多商业化的应用。在医学上，手术期和重症监护与治疗阶段，需要获取大量的信息，既有纯数据、又有生物信号、图像、文字等；既有数字化确定的信息，又有不确定和模糊的表述；既有静态的，又有动态的；既有共性的征象，又有个体差异，客观上为新技术的应用提供了广阔的舞台。

在信号处理、基于动态数据驱动的辅助决策专家系统、数据挖掘、各种临床状况的预测、智能化床旁监护、远程医疗与教学、医疗机器人等各方面运用神经网络技术和其他人工智能技术，帮助忙碌的医护人员更有效、安全、经济地为病人服务。

3. 图像处理

神经网络所具有的自学习，自适应和强大的信息综合能力，决定了它可以很好地模拟人眼的功能，完成普通的边缘算子难以完成的边缘检测。同时，神经网络边缘检测算法通过训练可以识别噪声与边缘图像的差异，从而去除噪声引起的虚假边缘，具有很好的抗噪性。

运用神经网络进行数字图像的边缘检测成为近年来的研究热点，涌现出大量的理论和方法，所用算法几乎涵盖了各种类型的神经网络。相信随着神经网络技术的不断革新，运用神经网络技术的图像处理技术必然也会得到迅速的发展。

4. 经济金融

自 20 世纪 80 年代末至今，人工智能得到了很大的发展，特别是神经网络的研究取得了划时代的发展。在经济金融领域，主要是应用神经网络方法进行投资预测，这是近年来的研究热点。

从 1988 年开始，人们利用 BP 网络对 IBM 股票进行预测，其目标是未来一个月内买进和卖出的最佳时机；从此以后，人们利用神经网络对世界各大股票市场的价格走势、各种证券指数的变动、投资组合决策进行了长短期的预测。神经网络的模式识别、自适应逼近，以及适应于非线性的能力获得了充分的应用。

6.4 神经网络工具箱简介

6.4.1 工具箱的功能

神经网络工具箱扩充了 MATLAB 在设计、应用、显示和仿真神经网络上的功能，可以将其用来解决常规难以解决的问题，并且可应用于各个领域，以实现各种复杂的功能。

神经网络工具箱由神经网络数据对象，各种类型的神经网络创建函数、传递函数、学习规则、图形用户界面、Simulink 工具函数等组成。因为神经网络需要复杂的矩阵计算，MATLAB 提供了一个神经网络数据结构，使得用户可以快速地学习和应用神经网络。

此外，神经网络工具箱还提供了方便用户设计和管理网络的图形用户界面（GUI），以及大量已经被证实的标准、开放、可扩展的设计实例。图形界面工具使得使用神经网络变得简单。它能够导入大量复杂的数据，并能够很快地产生、初始化、训练、仿真和管理网络。其中简单的图像表示有助于明确和理解网络的结构。

神经网络工具箱中包含了许多示例，每一个示例讲述一个问题。示例中展示了用来解决问题的网络，并给出最后的结果。

神经网络工具箱包含在 nnet 目录中，在 MATLAB 命令行中输入“help nnet”可得到神经网络工具箱的帮助主题。

MATLAB 神经网络工具箱提供了多种类型的神经网络模型，包括感知器网络、线性神经网络、BP 神经网络、径向基神经网络、反馈神经网络、自组织神经网络和学习矢量量化神经网络，等等，这些内容将在本书的后续章节中进行详细介绍。

这些神经网络模型可以应用于多个方面，主要包括：

（1）函数拟合与逼近。

（2）模式识别、图像处理。

（3）信息处理、时间序列预测。

（4）工业控制与故障诊断。

在实际应用中，首先需要根据问题的实际情况，对问题进行定义，同时确定输入/输出样本集；然后选择合适的网络模型，通过训练样本完成网络训练，使之适用于该实际问题的解决，从而实现模式分类、数据预测、函数拟合等功能；最后选择合理的测试样本，对训练后的网络进行仿真，检验网络的性能。

利用 MATLAB 神经网络工具箱，可以轻松地完成很多需要复杂烦琐的操作才能够完成的工作，有利于提高效率，减轻工作和研究人员的负担。

6.4.2 工具箱的新特性

MATLAB R2008b 中的神经网络工具箱版本号为 Version 6.0.1，其内容与 MATLAB R2008a 中的神经网络工具箱 6.0 版本基本一致。相对于此前的工具箱版本，6.0 及其之后的版本又增加了一些新的特性。主要的新特性和改动包括：

1．新的具有动画绘图功能的 GUI 训练工具

目前可以自动地在窗口中打开利用训练函数对网络进行训练的功能，此窗口将会显示网络结构框图、训练算法名称，以及训练状态信息。

此外，此窗口中同时包含了几个与被训练网络相关的按钮，通过这些按钮可以在训练中或训练后启动绘图程序。如果在训练过程中启动绘图，则绘图会随训练进度逐步更新，从而以动画的方式显示网络误差性能在训练过程中的变化。

应用下面的命令可以打开和关闭上述训练窗口：

- nntraintool
- nntraintool('close')

与绝大多数网络训练相关的绘图函数如下。

- plotperform：绘制误差性能。
- plottrainstate：绘制训练状态。

2．新的模式识别网络、绘图和分析 GUI 工具

利用函数 nprtool 可以打开一个 GUI 界面，此界面将引导用户解决模式识别问题。用户可以在此界面定义自己的问题，或者选择使用任意提供的数据集。

应用 newpr 函数可以以命令行的方式生成一个模式识别网络。这实际是一种前向型 BP 网络，适用于 2 选 1 或 N 选 1 的问题的求解。

3．新的聚类训练、初始化以及绘图 GUI 工具

利用 nctool 函数可以打开一个 GUI 界面，此界面可以引导用户生成用于解决聚类问题的自组织映射网络。用户可以定义自己的问题，也可以使用提供的数据集。

应用 initsompc 函数可以初始化自组织映射层的权值，以便加速训练过程。应用 learnsomb 函数可以实现对 SOM 网络的批处理训练，其训练速度比逐步训练有数量级上的提升。应用 newsom 函数可以应用快速算法创建一个 SOM 网络。

此外，与自组织映射网络相关的新的绘图函数还包括如下命令。

- plotsomhits：绘制自组织映射输入样本命中。
- plotsomnc：绘制自组织映射邻元连接。
- plotsomnd：绘制自组织映射邻元距离。
- plotsomplanes：绘制自组织映射输入权值平面。
- plotsompos：绘制自组织映射权值位置。
- plotsompos：绘制自组织映射拓扑。

4．新的网络结构查看器与改进的结构框图

新的神经网络结构框图支持绝对连接网络，同时具有更好的外观，通过色彩和阴影对视觉效果进行改进。网络结构框图出现在所有的神经网络图形界面工具中。此外，通过命令行方式也可以查看任意神经网络的结构。使用的命令如下：

```
view(new)
```

5. 新的拟合网络，绘图和更新的拟合 GUI 工具

应用 newfit 函数可以创建一个包含前向 BP 网络以及拟合绘图工具的拟合网络。神经网络工具箱也考虑了对以往版本的兼容性问题，尽管 nftool 生成的代码与以往版本不同，但在以往版本中编写的程序仍然能够在新的版本中正常运行。

6.4.3 MATLAB 中的神经网络数据结构

MATLAB 神经网络工具箱中，所创建的神经网络都通过一个经过定义的网络对象数据结构来表征。通过 MATLAB 提供的各种网络创建函数命令即可生成网络对象，同时也可以查看其数据结构的内容。

例如，输入如下简单的命令，就可以创建一个网络对象：

```
p=[0 1 0 1; 0 0 1 1],
t=[1 1 0 0],
net = newff(minmax(p), [2, 1], {' logsig', ' purelin'}, ' trainlm'),
```

从输出的语句中我们可以看到其基本的网络对象属性，此对象数据结构定义如下：

```
net =
    Neural Network object:
    architecture:
         numInputs: 1
         numLayers: 1
       biasConnect: [1]
      inputConnect: [1]
      layerConnect: [0]
     outputConnect: [1]
        numOutputs: 1  (read-only)
    numInputDelays: 0  (read-only)
    numLayerDelays: 0  (read-only)
    subobject structures:
            inputs: {1x1 cell} of inputs
            layers: {1x1 cell} of layers
           outputs: {1x1 cell} containing 1 output
            biases: {1x1 cell} containing 1 bias
      inputWeights: {1x1 cell} containing 1 input weight
      layerWeights: {1x1 cell} containing no layer weights
    functions:
          adaptFcn: 'trains'
         divideFcn: (none)
       gradientFcn: 'calcgrad'
           initFcn: 'initlay'
        performFcn: 'mae'
          plotFcns: {'plotperform','plottrainstate'}
          trainFcn: 'trainc'
    parameters:
        adaptParam: .passes
       divideParam: (none)
```

```
        gradientParam: (none)
            initParam: (none)
         performParam: (none)
           trainParam: .show, .showWindow, .showCommandLine, .epochs,
                      .goal, .time
    weight and bias values:
                   IW: {1x1 cell} containing 1 input weight matrix
                   LW: {1x1 cell} containing no layer weight matrices
                    b: {1x1 cell} containing 1 bias vector
    other:
                 name: ''
             userdata: (user information)
```

上面给出了网络对象包含的属性，其中各组属性分别介绍如下。

1. 网络结构属性（architecture）。

（1）numInputs：网络输入数目。

（2）numLayers：网络的层数。

（3）biasConnect：网络偏差连接属性。

（4）inputConnect：输入连接属性。

（5）layerConnect：层连接属性。

（6）outputConnect：输出连接属性。

（7）numOutputs：输出向量数目。

（8）numInputDelays：输入延迟属性。

（9）numLayerDelays：层延迟属性。

需要指出的是，参数中网络输入数目与网络输入向量大小是不一样的。前者由参数 numInputs 决定，而后者由参数 net.inputs{i}.size 决定。

2. 子对象属性（subobject structures）。所谓子对象属性，是指定义网络输入、层、输出、目标、权值阈值向量等的元胞矩阵。

（1）inputs：输入向量。

（2）layers：网络层。

（3）outputs：输出向量。

（4）biases：偏差向量。

（5）inputWeights：输入权值。

（6）layerWeights：层权值。

3. 网络函数（functions）。

（1）adaptFcn：自适应训练函数。

（2）devideFcn：网络训练过程中选用的数据划分函数。

（3）gradientFcn：梯度函数。

（4）initFcn：初始化函数。

（5）performFcn：误差性能函数。

（6）plotFcns：绘图函数。

（7）trainFcn：训练函数。

4．网络参数（parameters）。

（1）adaptParam：自适应训练函数参数。

（2）divideParam：数据分组参数。

（3）gradientParam：梯度函数参数。

（4）initParam：初始化函数参数。

（5）performParam：误差性能函数参数。

（6）trainParam：训练函数参数。

5．权值和偏差值（weight and bias values），即权值和偏差矩阵的值。

（1）IW：输入权值矩阵。

（2）LW：层权值矩阵。

（3）b：偏差矩阵。

6．其他属性。

net.userdata：用户增加的自定义信息

需要指出的是，以上列出的许多网络属性之间会相互影响，改变其中的一个值，其他与之相关的参数随之变动，例如对网络输入数目 numInputs 的改动同时会引起输入连接属性 inputConnect 参数和子对象参数 net.inputs 的变化；对 net.biasConnect 参数的改动会引起其他参数如 net.biases 以及 net.b 的变动。

6.4.4 工具箱函数简介

MATLAB 神经网络工具箱为用户提供的函数大体可分为两类：第一类是通用函数，大体上对所有网络都能够应用，例如神经网络的初始化函数、仿真函数，以及训练函数等；第二类则是某一类型的神经网络专用的，例如各种类型网络的创建函数等。

这些函数将设计网络过程中的复杂的计算进行了封装，这使得设计者可以从网络设计的繁重工作中跳出来，专注于问题的解决方法的思考。表 6-1 列出了主要的工具箱函数及其说明，函数的具体应用将在后续章节中介绍。

表 6-1　神经网络工具箱常用函数列表

通用函数			
train	训练函数	hardlims	对称硬限值函数
trainb	权值阈值训练函数	init	网络初始化
adapt	自适应训练函数	initlay	多层网络初始化
learnp	网络学习函数	initnw	用 NW 规则初始化
learnpn	标准学习函数	initwb	对指定层初始化
sim	神经网络仿真函数	netsum	对输入求和
hardlim	硬限值函数	netprod	对输入求积

续表

专用函数			
newp	新建感知器网络	plotpv	绘制感知器目标向量
newlin	新建线性网络	plotpc	绘制感知器决策分界线
newlind	设计一个线性网络	learnwh	Widrow-Hoff 学习函数
newff	新建前馈网络	maxlinlr	计算最大线性学习速率
newffd	新建延迟前馈网络	logsig	S 对数传递函数
newlrn	新建层回馈网络	dlogsig	logsig 导数函数
newhop	新建 Hopfield 网络	tansig	S 正切传递函数
newrb	新建 RBF 网络	dtansig	tansig 导数函数
newrbe	新建一个严格的 RBF 网络	purelin	线性函数
newpnn	新建概率神经网络	learngd	梯度下降学习函数
newgrnn	新建广义回归网络	learngdm	带动量梯度下降学习函数
newc	新建竞争层	radbas	径向基传递函数
newsom	新建自组织映射网络	compet	竞争传递函数
newlvq	新建学习矢量量化网络	softmax	软最大传递函数

在表 6-1 中，通用函数主要是网络训练、初始化、仿真等函数，专用函数则主要是各种网络创建函数、网络传递函数、训练算法函数等。对于它们的具体调用格式和应用实例，将在后续的章节中根据内容进行详细介绍。

6.5 小结

MATLAB 神经网络工具箱充分利用了计算机所提供的软硬件资源和先进思想，功能全面、应用便捷，是一个非常优秀的神经网络工具软件包，在国外，它已经得到了很好的推广和应用。精通 MATLAB 神经网络工具箱的使用，将使得设计者如虎添翼，腾出更多时间思考和研究出功能更强、更加有效的神经网络和应用成果。



第 7 章 MATLAB 神经网络 GUI 工具

为方便用户更直观更容易地应用神经网络进行数据分析，MATLAB 提供了神经网络图形用户界面（Neural Network Graphic User Interface）的功能，该图形界面软件是神经网络工具上的一个补充，具有简洁、友好的用户接口，可以方便地应用于实际的数据分析，主要适用于函数拟合、模式识别、数据聚类这几个主要方面。

7.1 基础 GUI 工具 nntool

要应用神经网络图形用户界面工具，首先需要在 MATLAB 中应用 nntool 命令调出 Network/Data 管理器窗口，此窗口具有自己的工作区，是与 MATLAB 自身的命令行工作区分开的。因此，使用 GUI 的时候，需要通过操作将 GUI 的结果导出到 MATLAB 工作区。同时，也可能需要将 MATLAB 工作区的执行结果导入到 GUI 中。

当打开 Network/Data 管理器窗口之后，就可以利用它创建、查看、训练神经网络，当然也可以进行仿真和数据处理，然后将处理得到的结果导出到 MATLAB 工作区中。

本章我们将以实例的形式，讲解如何利用神经网络 GUI 生成一个感知器神经网络，并进行数据处理，与 MATLAB 工作区进行数据导入导出的交互过程。主要内容包括：

- 创建网络。
- 训练网络。
- 网络仿真。
- 数据导出。
- 数据清除。
- 数据导入。
- 数据保存与读取。

下面各节利用 GUI 创建一个具有逻辑与功能的感知器神经网络，并进行训练和仿真。

7.1.1 网络创建

本节介绍如何利用 GUI 创建所需要的神经网络。

【例 7-1】 利用 GUI 以及期望输入 ***p***= [0 0 1 1;0 1 0 1]，期望响应 ***t***=[0 0 0 1]，创建一个具有逻辑与功能的感知器神经网络。

解： ① 首先启动 GUI，在 MATLAB 工作区输入命令：

```
nntool
```

此命令打开 Network/Data 管理器窗口，其界面如图 7-1 所示。

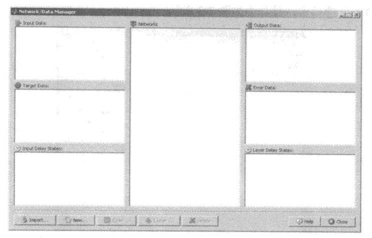


图 7-1 Network/Data 管理器窗口

从图中可见，Network/Data 管理器窗口分为 7 个区域，包含 7 个按钮。

（1）“Input Data” 区域：用于显示用户指定的输入向量。

（2）“Target Data” 区域：用于显示用户指定的期望目标响应。

（3）“Input Delay States” 区域：用于显示设置的输入延迟情况。

（4）“Networks” 区域：用于显示操作中网络的设置。

（5）“Output Data” 区域：用于显示网络的仿真输出。

（6）“Error Data” 区域：用于显示网络的学习误差。

（7）“Layer Delay States” 区域：用于显示网络各层的延迟状态。

操作按钮的名称与功能分别如下。

（1）“Import” 按钮：从 MATLAB 工作区或磁盘导入数据。

（2）“New” 按钮：生成新的网络或数据。

（3）“Open” 按钮：打开网络或数据，实现查看、编辑、训练、仿真等操作。

（4）“Export” 按钮：将 GUI 数据导出到 MATLAB 工作区或磁盘中。

（5）“Delete” 按钮：删除选择的数据或网络。

（6）“Help” 按钮：获取 GUI 帮助。

（7）“Close” 按钮：关闭 GUI 窗口。

如果用户对 GUI 窗口不太熟悉，可以单击 “Help” 按钮。此时将弹出帮助窗口，为用户介绍 GUI 的使用步骤，以及上述区域和按钮的功能。

对于本例，首先需要指定创建网络的两维输入向量。单击 “New” 按钮，将弹出一个 Create Network or Data 窗口，生成这些需要的数据。

此窗口包含两个选项卡，定义和功能如下：

（1）“Network” 选项卡，用于定义新的网络对象。

（2）“Data”选项卡，用于定义新的数据。

单击“Data”选项卡，其界面如图 7-2 所示。

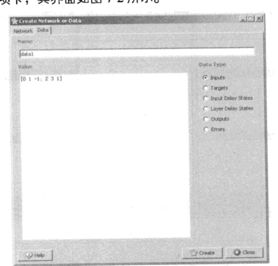


图 7-2 “Create Network or Data”窗口-“Data”选项卡

从图中可以看到，该选项卡包含如下两个区域。

（1）“Name”区域：用于定义新数据名。

（2）“Value”区域：用于给数据赋值。

选项卡的右部是“Data Type”单选按钮区，利用这些单选按钮可以定义不同的数据类型。

（1）Inputs：输入向量。

（2）Targets：期望响应向量。

（3）Input Delay States：输入延迟状态向量。

（4）Layer Delay States：层延迟状态向量。

（5）Output：输出向量。

（6）Errors：误差向量。

在此选项卡中，将“Name”区域的值设定为“p”，将 Value 的值设定为[0 0 1 1;0 1 0 1]，并将“Data Type”设定为“Inputs”。

单击“Create”按钮，在弹出的对话框中单击“OK”按钮，就创建了一个输入向量 p，重新弹出 Network/Data 管理器窗口，在“Input Data”区域中，将 p 显示为输入向量，如图 7-3 所示。

此时如果需要删除该向量，只需在窗口中选中“p”，然后单击“Delete”按钮，就可以删除该向量。如果想要编辑该向量，可以选中“p”，双击，此时会弹出数据编辑窗口，如图 7-4 所示，在此窗口中，可以更改向量 p 的值。

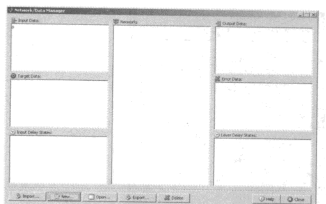


图 7-3　创建输入“p”后的 Network/Data 管理器窗口

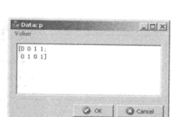


图 7-4　向量 ***p*** 的编辑窗口

以同样的方法定义网络期望的输出响应，在“Create Network or Data”窗口中，设置“Name”为“t”，“Value”区域的值为[0 0 0 1]，并选择“Target”作为“Data Type”，单击“Create”和“OK”按钮，就可以完成期望输出响应 ***t*** 的定义。

完成输入与期望响应向量定义后的 Network/Data 管理器窗口如图 7-5 所示。

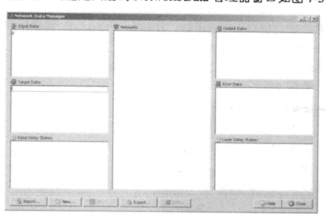


图 7-5　创建输入与期望响应后的 Network/Data 管理器窗口

② 利用图形用户界面创建该感知器网络。此神经网络名称定为 ANDNet，在 Network/Data 管理器窗口中单击“New”按钮，选择“Network”选项卡，该选项卡界面如图 7-6 所示。

图 7-6 中，默认的网络类型为 BP 网络，选项卡中的区域和功能如下。

（1）“Name”区域：定义网络名称。

（2）“Network Type”区域：定义网络类型。

（3）网络设置区域：可以对网络输入（Input data）、目标响应（Target data）、训练函数（Training function）、网络学习函数（Adaption learning function），误差性能（Performance function）函数和网络层数（Number of layers）进行设置。

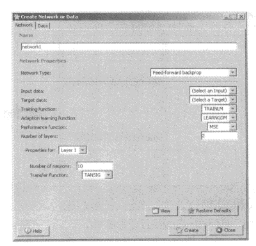


图 7-6 “Create Network or Data” 窗口- “Network” 选项卡

（4）层属性设置区域：对指定的某一层属性进行设置。

在本例中设置网络名称为“ANDNet”，选择网络类型为“Perceptron”，则窗口更换到“Network”选项卡。将网络输入指定为“p”，将网络输出指定为“t”，感知器采用硬限值函数作为传递函数，因此将网络传递函数“Transfer Function”选项指定为“HARLIM”。学习函数“Learning function”选项设置为“LEARNP”。

设置完后的 Network 选项卡窗口如图 7-7 所示。如果想查看此网络，可以单击“View”按钮，弹出的窗口如图 7-8 所示。

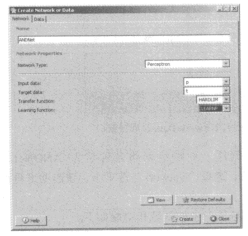


图 7-7 感知器网络设置完成后的“Network”选项卡

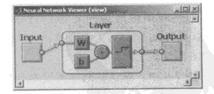


图 7-8 生成的感知器网络示意图

从图 7-8 中可以看出，我们期望建立的网络是一个单层网络，只包含单输入单输出（输入向量为二维向量）、单神经元结构，采用的传递函数为硬限值函数，这就是期望的感知器神经网络。

回到 7-7 图中所示的“Network”选项卡界面中，单击“Create”按钮，在弹出的对话框中单击“OK”按钮，即可以生成上述网络。随后可以单击“Close”按钮关闭“Create Network or Data”窗口，回到 Network/Data 管理器窗口。在其中，可以看到新生成的 ANDNet 出现在 Network 区域中，如图 7-9 所示。

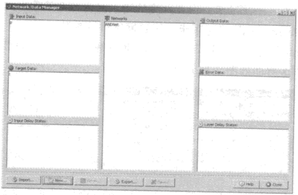


图 7-9 创建感知器后的 Network/Data 管理器窗口

在此对话框中双击“ANDNet”或者选中它然后单击“Open”按钮，就可以打开此生成的网络进行各种编辑、训练、仿真等操作。

7.1.2 网络训练

本节介绍如何利用 GUI 图形操作界面对 7.1.1 节中生成的网络进行训练。

为了对在 7.1.1 节中步骤②中生成的网络进行训练，首先在 Network/Data 管理器中单击“ANDNet”，选中该网络对象，然后单击“Open”按钮，将会弹出一个新窗口，窗口标题栏为“Network:ANDNet”，如图 7-10 所示。

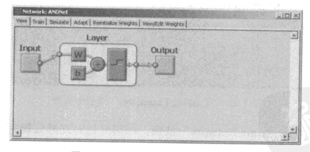


图 7-10 “Network:ANDNet”窗口

在此窗口中包含有 6 个选项卡，其定义和功能分别如下。

（1）“View”选项卡：用于查看选中的网络。

（2）“Train”选项卡：用于训练网络。

（3）“Simulate”选项卡：用于对网络进行仿真。

（4）“Adapt”选项卡：用于网络的自适应训练。

（5）“Reinitialize Weights”选项卡：用于对网络权值重新初始化。

（6）“View/Edit Weights”选项卡：用于查看和编辑网络权值。

为对网络进行训练，单击“Train”选项卡，其中包含两个子选项卡。

（1）“Training Info”子选项卡：设置网络的输入样本与期望输出。

（2）“Training Parameters”子选项卡：设置网络训练参数。

在“Training Info”选项卡中选定“p”作为网络输入，“t”作为目标响应，选定后的“Training Info”选项卡界面如图 7-11 所示。

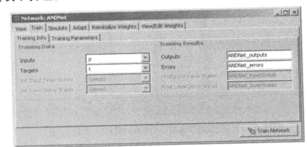


图 7-11 “Training Info”选项卡界面

注意，图中“Training Results”区域的“Outputs”和“Errors”的结果都含有“ANDNet_”前缀，这样做的目的是为了使得将训练结果数据导出到 MATLAB 工作区后更加容易识别。

单击“Training Parameters”选项卡，在此选项卡界面中，可以对网络训练最大迭代次数、误差性能目标进行设置，该选项卡界面如图 7-12 所示。

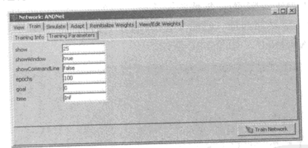


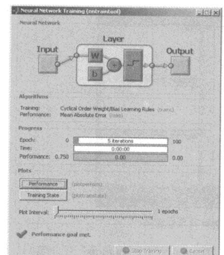
图 7-12 “Training Parameters”选项卡界面

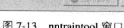
单击窗口右下角的“Train Network”按钮，即可开始对此感知器网络进行训练，训练过程中将会弹出如图 7-13 所示的 nntraintool 窗口。

这也正是在调用 train 函数对网络进行训练时所弹出的窗口，窗口中给出了网络的结构示意、训练采用的算法、误差性能函数，以及训练过程的动态显示。可以看到，经过 5 次迭代，网络就完成了训练，对于感知器神经网络，训练的结果能够达到零误差，而对于其他类型的神经网络则不然。

单击图 7-13 所示窗口中的“Performance”按钮，即可查看训练过程中网络误差性能

的变化情况，如图 7-14 所示。




图 7-13 nntraintool 窗口

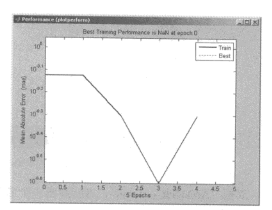


图 7-14 训练过程中误差性能的变化

注意，图中的纵坐标以对数的形式画出。

7.1.3 网络仿真

本节介绍利用 GUI 图形操作界面对前面生成的感知器神经网络进行仿真的过程。

下面利用输入向量 ***p*** 对训练后的网络对象 ANDNet 进行仿真，查看输出是否与期望目标响应相同。

在“Network:ANDNet”窗口中单击“Simulate”选项卡，将其中的“Inputs”项设置为“p”，如图 7-15 所示。在“Simulation Results”区域中的各项同样也有前缀“ANDNet_”，以清晰地显示变量所属网络对象。

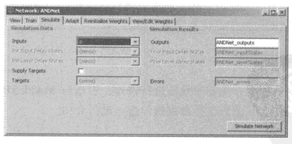


图 7-15 “Simulate”选项卡界面

单击右下角的“Simulate Network”按钮，在弹出的对话框中单击“OK”按钮，就可以对网络进行仿真了。

仿真完成之后，在 Network/Data 管理器中，在“Output_Data”区域会产生一个新的变

量，名为“ANDNet_outputs”，如图 7-16 所示。这就是网络仿真得到的输出结果。

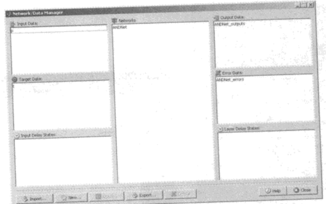


图 7-16　仿真后的 Network/Data 管理器窗口

双击“ANDNet_outputs”，将会弹出一个名为“Data: ANDNet_outputs”的小窗口，如图 7-17 所示。

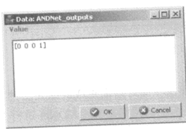


图 7-17　仿真结果窗口

从图中可以看到，仿真结果 ANDNet_outputs 的值为[0 0 0 1]，确实是我们期望的目标响应，这说明经过训练的网络实现了零误差输出。只在两个输入均为 1 的情况下输出才为 1，其他情况输出均为 0，这说明了网络具有逻辑与的功能。

单击“OK”按钮即可关闭此仿真结果窗口。

7.1.4　图形界面数据操作

在神经网络图形界面中，需要对数据进行操作，包括从 MATLAB 工作区导入、导出数据、清除数据等操作。下面对这些数据操作逐一进行介绍。

1．导出数据

接续 7.1.3 的步骤，将感知器网络的仿真结果导出到 MATLAB 工作区中。

回到 Network/Data 管理器窗口，ANDNet 网络仿真的输出变量和误差向量分别列在窗口右侧的“Outputs”区域和“Errors”区域。

单击“Export”按钮，将会弹出一个“Export from Network/Data Manager”对话框，其

中列出了将要导出的各个变量，此例中的变量包括 ***p***，***t***，ANDNet，ANDNet_outputs，ANDNet_errors，如图 7-18 所示。

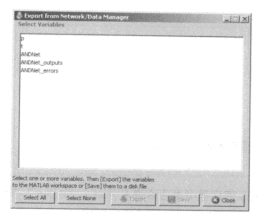


图 7-18 “Export from Network/Data Manager”窗口

在此窗口中，除了“Select Variables”变量区域之外，还包含如下 5 个按钮。

（1）“Select All”按钮：全选输出变量。

（2）“Select None”按钮：清除所有选择。

（3）“Export”按钮：导出到 MATLAB 工作区。

（4）“Save”按钮：保存到磁盘文件。

（5）“Close”按钮：关闭窗口。

单击选择变量 ANDNet_outputs，ANDNet_errors，然后单击“Export”按钮，在弹出的对话框中单击“OK”按钮，即可将仿真结果导出到 MATLAB 工作区中。

我们可以回到 MATLAB 工作区中验证导出操作的结果，输入命令：

```
ANDNet_outputs
```

输出结果为：

```
ANDNet_outputs =     0     0     0     1
```

输入命令：

```
ANDNet_errors
```

输出结果为：

```
ANDNet_errors =     0     0     0     0
```

验证了导出操作确实是成功完成的。

采用同样的方式，也可以将变量 ***p***，***t***，ANDNet 导出到 MATLAB 工作区中，而应用同样的方法也可以验证导出的结果。

一旦网络对象 ANDNet 被导出到了 MATLAB 工作区，我们就可以通过命令行的方式对网络的参数进行查看。例如，输入命令：

```
ANDNet.iw{1,1}
```

则输出结果为：

```
ans =     2     1
```

输入命令：

```
ANDNet.b{1}
```

则输出结果为：

```
ans =    -3
```

以上就是训练后的感知器网络的权值和偏差。

2．清除数据

通过在 Network/Data 管理器中单击变量名并单击“Delete”按钮，即可以将变量删除，在所有的变量删除完毕后，可以获得一个清空状态的 Network/Data 管理器。

清除数据的另一个方法是，退出 MATLAB 程序，然后重新启动 MATLAB，在 MATLAB 命令行中输入 nntool 命令，重新启动 Network/Data 管理器，这样也可以得到一个清空状态的 Network/Data 管理器。

需要指出的是，如上面实例所述，如果将 ***p***，***t*** 等变量导出到了 MATLAB 工作区中，那么即使关闭了 Network/Data 管理器，这些变量仍然存在于 MATLAB 工作空间，不会随 Network/Data 管理器的关闭而被清除。

3．从工作区中导入数据

下面介绍怎样从 MATLAB 工作区中将数据导入 GUI 中。

为了方便后面的处理，首先关闭 Network/Data 管理器，再输入 nntool 命令重新启动图形操作界面。此时的 Network/Data 管理器中不含有任何变量。

在 MATLAB 工作区中定义一个变量，输入命令：

```
r= [0; 1; 2; 3]
```

得到：

```
r = 0
    1
    2
    3
```

接下来回到 Network/Data 管理器，单击“Import”按钮，弹出一个新窗口，如图 7-19 所示。

此窗口中包含三个区域。

（1）“Source”区域：选择从 MATLAB 工作区或从磁盘文件中导入数据。

（2）“Select a Variable”区域：选择需要导入的变量名。

（3）“Destination”区域：选择导入后的变量名和类型。

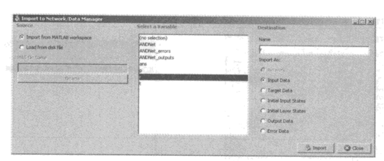


图 7-19 “Import to Network/Data Manager” 窗口

在 “Source” 区域中选择 “Import from MATLAB workspace”，在 “Select a Variable” 区域选择变量 “r”，此时 “Name” 也自动变为 “r”，选择 “Import As” 为 “Input Data”，然后单击右下角的“Import”按钮，在弹出的窗口中单击“r”，就可以看到新变量 r 从 MATLAB 工作区导入了 GUI 中。导入数据后的 Network/Data 管理器窗口如图 7-20 所示。

在此窗口中，双击 “r” 可以查看其内容，弹出的窗口如图 7-21 所示。

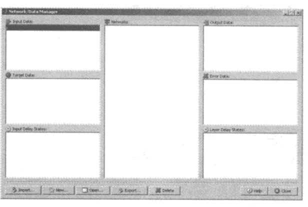


图 7-20 导入数据后的 Network/Data 管理器窗口

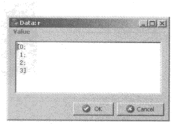


图 7-21 查看导入数据内容

可以看到，数据确实无误地被导入到了 GUI 空间中。

4. 数据保存与读取

下面介绍如何对数据进行保存和读取操作。

首先打开 Network/Data 管理器，随后单击 “New Network” 按钮，将新的网络名称设置为 “mynet”，选择网络类型为竞争性网络，选择前面提过的 “r” 作为输入向量，如图 7-22 所示。

单击 “Create” 按钮，在弹出的对话框中单击 “OK” 按钮，于是就创建了名为 “mynet” 的网络对象，并且该对象出现在 Network/Data 管理器的 Networks 区域中。

随后将其导出到磁盘文件。在 Network/Data 管理器中，单击“Export”按钮，再在 Export 窗口的 “Select Variables” 区域选择 “mynet” 作为导出对象，如图 7-23 所示。

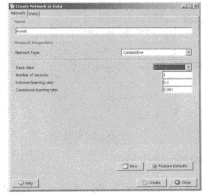


图 7-22　创建“mynet”网络对象

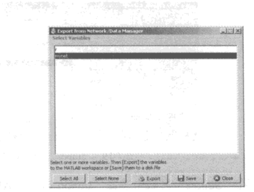


图 7-23　选择导出对象

在此窗口中，单击“Save”按钮即可将其保存为磁盘文件，将文件名命名为“mynet”，如图 7-24 所示，格式为 MAT 文件。以上就是将 GUI 中变量保存为磁盘文件的过程。



图 7-24　保存 mynet 网络对象

下面将 GUI 中的变量删除，然后再通过文件导入回来。

首先回到 Network/ Data 管理器，单击“mynet”变量，再单击“Delete”按钮，就将此变量从空间中删除。随后单击“Import”按钮，在弹出的 Import 窗口中选择“Load from Disk File”，然后利用“Browse”按钮找到刚才保存的 mynet.MAT 文件，将其导入即可，如图 7-25 所示。

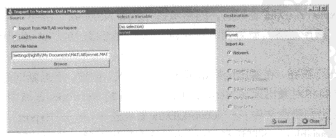


图 7-25　从磁盘文件导入 mynet 网络对象

单击“mynet”，然后单击“Load”按钮，就可以将此变量重新导入到 Network/ Data 管理器窗口中，读入数据后的 Network/ Data 管理器窗口如图 7-26 所示。

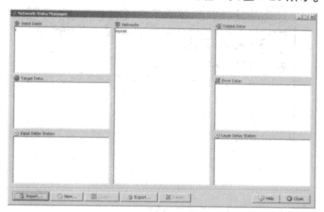


图 7-26 读入数据后的 Network/ Data 管理器窗口

从图中可见，“mynet”变量又重新出现在“Networks”区域中。

7.2 数据拟合 GUI 工具 nftool

上面介绍了基础 GUI 工具 nntool 的使用。除此之外，MATLAB 神经网络工具箱还为函数拟合、模式识别、数据聚类方面的应用，提供了几个专门的 GUI 工具，分别是 nftool、nprtool、nctool。其中函数拟合 GUI 工具 nftool 我们在第 2 章中已经进行了介绍，这里将对其他两个工具的应用进行实例介绍。

神经网络的一个非常重要的应用方向就是函数拟合，利用非常简单的网络就可以实现对实际问题中较复杂的数据关系的拟合。对于一个实际的函数拟合问题，在 MATLAB 神经网络工具箱框架下，用户可以选择如下三种方式进行处理：

（1）应用命令行方式创建网络进行处理，这种方式是我们后续章节中主要介绍的。

（2）应用神经网络拟合 GUI 工具 nftool。

（3）应用基本 GUI 工具 nntool 进行处理。

在此，我们选择 nftool 进行实例分析。

【例 7-2】 nftool 分析实例。假定有一个房屋价格问题，现有 506 个住房的 13 组相关数据构成的样本数据集，该数据集由 MATLAB 以示例数据的形式提供，利用 nftool 工具，设计一个神经网络，对住房市场价格进行拟合分析。

解： ① 首先调用 nftool 命令，打开拟合图形界面窗口，输入命令：

```
nftool
```

打开的 nftool 窗口如图 7-27 所示。

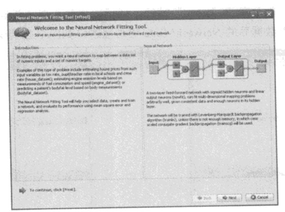


图 7-27　nftool GUI 工具窗口

② 单击“Next”按钮，进入到数据选择窗口，如图 7-28 所示。

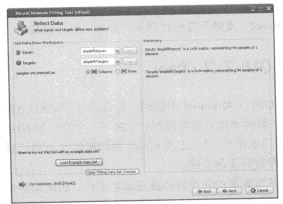


图 7-28　nftool 数据选择窗口

在此窗口中，可以通过三种方式选择网络训练的样本数据。可以单击窗口中的“Inputs”和“Targets”下拉列表框从 MATLAB 工作空间中选取样本数据，也可单击后面的“...”按钮从文件中导入数据，此外，也可单击“Load Example Data Set”按钮，导入 MATLAB 提供的实例样本数据。

此处，我们选择 MATLAB 提供的样本数据，单击“Load Example Data Set”按钮，选择“House Pricing”数据集选项，此数据集共有 506 个住房数据，数据为 13×506 的矩阵，各行定义分别为：

（1）城镇犯罪率。

（2）城镇面积在 25000 平方英尺以上的居住面积比例。

（3）城镇非零售商业区比例。

（4）是否临河。

（5）氮氧浓度（ppm）。

（6）居民平均房间数。

（7）1940 年之前的住房单元比例。

（8）到 5 个商业中心的加权距离。

（9）高速公路指数。

（10）税率。

（11）小学受教育率。

（12）$1000(Bk-0.63)^2$，Bk 为城镇黑人数。

（13）低生活质量居民人口百分比。

单击“Import”按钮，导入数据，回到图 7-28 所示界面。

③ 单击“Next”按钮，进入到验证与测试数据窗口，如图 7-29 所示。

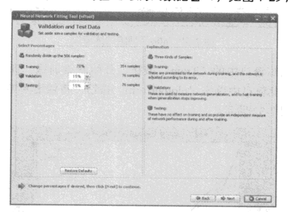


图 7-29 nftool 验证与测试数据窗口

在此窗口中，可以设定用来进行训练的样本数据、验证数据，以及测试数据比例。可根据默认的比例设定，应用样本数据的 70%进行训练，15%进行验证，15%进行测试。

④ 单击“Next”按钮，进入网络尺寸设置窗口，如图 7-30 所示。设置网络尺寸，即隐层神经元数目为默认的 20 个神经元。

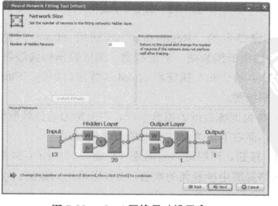


图 7-30 nftool 网络尺寸设置窗口

⑤ 单击“Next”按钮，进入网络训练窗口，如图 7-31 所示。

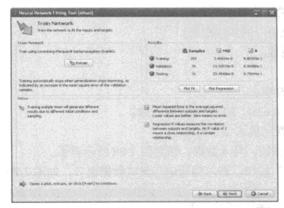


图 7-31　nftool 网络训练窗口

⑥ 单击网络训练窗口中的“Train”按钮，开始网络训练，训练过程中会自动弹出 nntraintool 窗口，在此窗口中单击“plotperform”按钮，将绘出网络误差性能随训练而变化的曲线，曲线如图 7-32 所示。

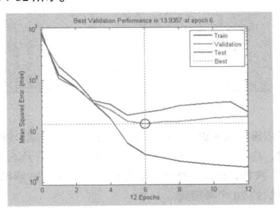


图 7-32　网络性能变化曲线

单击 nntraintool 窗口中的“Regression”按钮或 nftool 网络训练窗口中的“Plot Regression”按钮，将绘出训练数据、验证数据、测试数据的曲线回归情况，如图 7-33 所示。而 nntraintool 窗口中的“Fit”按钮和 nftool 训练窗口中的“Plot Fit”按钮，则只能用于单输入单输出的回归绘图。

图 7-33 中，左上角为训练数据的拟合情况，右上角为验证数据的拟合情况，左下角为测试数据的拟合情况，而右下角为全体数据的拟合情况。

⑦ 单击“Next”按钮，可以进入到网络评估窗口，如图 7-34 所示。在此窗口中，可以通过增加或减少网络隐层中神经元的数目，或者应用新的数据、对创建的网络进行评估，这里不再赘述。

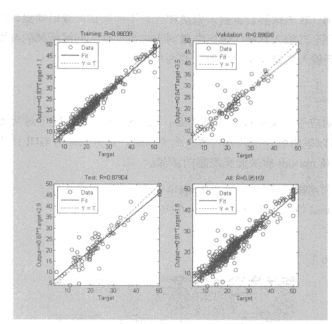


图 7-33 数据回归情况

接下来单击“Next”按钮，进入到结果保存窗口，如图 7-35 所示。在此窗口中可以将生成的网络等数据保存到 MATLAB 工作空间，或者产生相应的 Simulink 框图。为要保存的变量命名，然后单击“Finish”按钮，即可完成此曲线拟合过程。

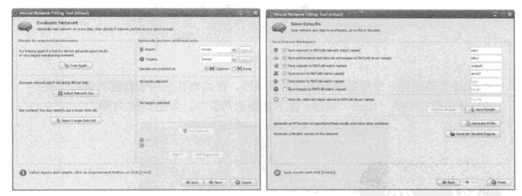


图 7-34 nftool 网络评估窗口　　　　图 7-35 nftool 结果保存窗口

7.3 模式识别 GUI 工具 nprtool

模式识别是神经网络的重要应用方向，对此，MATLAB 神经网络工具箱提供了专门的 nprtool 模式识别 GUI 工具，利用此工具，我们可以快速地进行模式识别的应用和学习，下面对其进行介绍。

对于模式识别，首先需要设置输入向量和代表不同模式的输出向量，每个输入向量以列向量的方式排列形成输入网络，输出向量则代表每个输入向量所属的类别。例如下列语句定义了四个二元输入向量及其分别对应的两个类别:

```
inputs = [0 1 0 1; 0 0 1 1];
targets = [0 1 0 1];
```

上面的问题对应于两类的模式识别问题，根据输出为 1/0 即可对两类进行划分。在 MATLAB 中既可以采用命令行的方式解决此类问题，也可以采用 GUI 的方式，下面我们介绍应用 GUI 工具 nprtool 解决此类问题的步骤。

【例 7-2】 利用模式识别 GUI 工具 nprtool 进行模式识别练习。

解: ① 输入 nprtool 命令，打开模式识别 GUI 工具:

```
nprtool
```

弹出如图 7-36 所示的窗口。

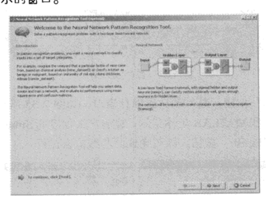


图 7-36 nprtool 窗口

此 GUI 工具生成一个两层的前向型神经网络，在此将其应用于数据的模式识别。

② 单击窗口中的“Next”按钮，进入数据选择窗口，如图 7-37 所示。

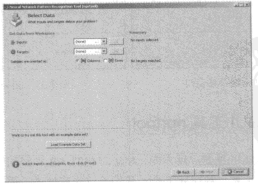


图 7-37 nprtool 数据选择窗口

单击“Load Example Data Set”按钮，调用 MATLAB 提供的样本数据，弹出模式识别数据集选择子窗口，如图 7-38 所示。

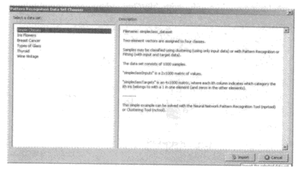


图 7-38　nprtool 数据集选择子窗口

在此窗口中选择“Simple Classes”，然后单击“Import”按钮，将导入此简单分类样本数据，并回到图 7-37 的数据选择窗口。

此样本数据集为一组二元输入向量，并将其划归为 4 个类别，共含有 1000 个样本数据，即输入样本向量为一个 2×1000 矩阵，输出样本向量为一个 4×1000 矩阵。输出矩阵每一列为一个四元向量，元素取值为 0 或 1，分别表征四个类别的隶属情况，为 1 表示属于该类别，为 0 则表示不属于该类别，如[0;1;0;0]，表示输入向量属于类别 2。

③ 单击“Next”按钮，进入到验证与测试数据窗口，如图 7-39 所示。

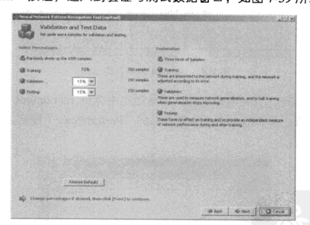


图 7-39　nprtool 验证与测试数据窗口

选择将样本数据中的 15%分别作为验证数据和测试数据。

④ 单击“Next”按钮，进入到网络尺寸设置窗口，如图 7-40 所示。

默认网络隐层神经元数目为 20，用户可以将其设置为任意合适的值，这里我们采用默认值。

⑤ 单击“Next”按钮，将生成对应的网络，并进入到网络训练窗口，如图 7-41 所示。

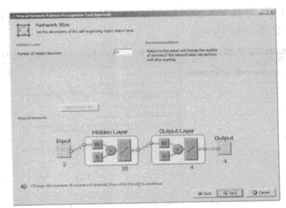


图 7-40　nprtool 网络尺寸设置窗口

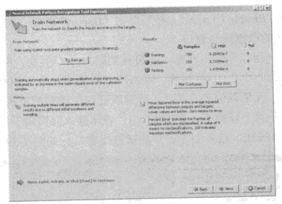


图 7-41　nprtool 网络训练窗口

单击“Train”按钮，开始网络训练，训练过程中弹出 nntraintool 窗口，此处经过 33 次迭代完成训练过程。单击 nntraintool 窗口上的“Performance”按钮，将弹出网络误差曲线，如图 7-42 所示。

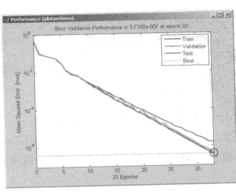


图 7-42　训练过程中的网络误差曲线

⑥ 单击 nntraintool 窗口上的“Confusion”或 nprtool 网络训练窗口上的“Plot Confusion”按钮，可以绘出网络对训练数据、验证数据、测试数据的模式识别效果图，如图 7-43 所示。

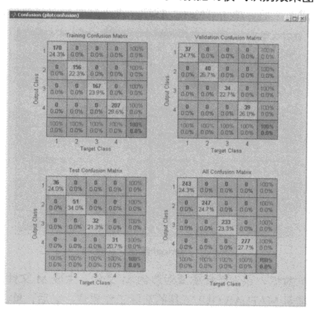


图 7-43 模式识别效果图

图 7-43 中，左上角、右上角、左下角、右下角分别为训练数据、验证数据、测试数据，以及全体样本数据的识别效果。其中，绿色方格代表各组数据中各类的真实比例情况，红色方框代表各类之间错误划分的情况，灰色方框内的数据为每一类数据的划分正确率与错误比率，蓝色方框代表了整体的划分正确率。可以看到网络对任意一组数据的识别效果，红色方框内的比例均为 0%，正确率为 100%，所以网络的划分效果几乎是完美的。

⑦ 单击 nprtool 网络训练窗口上的 “Next” 按钮，进入到网络评估窗口，如图 7-44 所示。

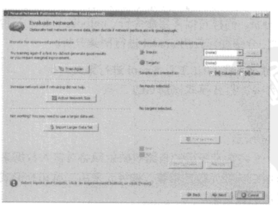


图 7-44 nprtool 网络评估窗口

在此窗口下，可以利用新数据对网络进行检验，如果对网络效果不满意的话，可以重新训练，改变神经元数目或者采用新的训练数据集。

在对网络性能满意之后，单击“Next”按钮，进入到数据保存窗口，如图 7-45 所示。

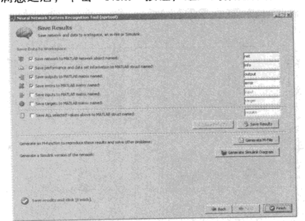


图 7-45 nprtool 数据保存窗口

选择希望保存的数据，或者选择生成对应的 M 文件，生成 M 文件是一个非常有用的选项，有助于我们理解 MATLAB GUI 工具的实际工作原理。之后，单击“Finish”按钮即完成此次 nprtool GUI 工具的模式识别。

7.4 数据聚类 GUI 工具 nctool

对数据进行聚类分析是神经网络一个非常优秀的功能。所谓数据聚类，是指通过数据的相似度对数据进行归档。聚类操作可能的应用方向有：

- 通过对人们的消费偏好分析进行市场划分。
- 对数据进行数据挖掘和子集划分。
- 通过对基因表达形式归类进行生物信息学研究。

对数据进行聚类分析，同样可以采用命令行、nntool、nctool 三种方式，其中后两种均为 GUI 方式，这里我们介绍如何应用 nctool 进行聚类分析。

【例 7-3】 利用数据聚类 GUI 工具 nctool 进行聚类分析。

解：① 输入 nctool 命令，打开模式识别 GUI 工具：

```
nctool
```

弹出窗口如图 7-46 所示。

nctool 通过创建一个包含竞争层的自组织映射网络对输入数据集进行聚类分析，它能够表征的维数与其自组织神经元数目相等，神经元采用二维拓扑形式排列，因而能够表征具有二维拓扑分布性质的数据集。

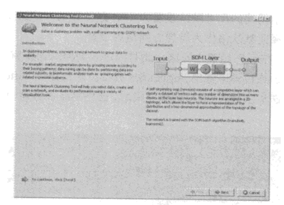


图 7-46 nctool 数据聚类 GUI 工具

② 单击“Next”按钮，弹出数据选择窗口，此过程与 nprtool 工具的基本相同。在此窗口中单击“Load Example Data Set”按钮，选择使用 MATLAB 提供的样本数据集，弹出数据集选择窗口，如图 7-47 所示。

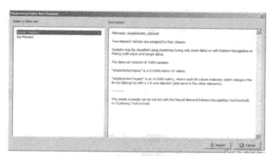


图 7-47 nctool 数据集选择窗口

选择“Simple Clusters”，然后单击“Import”按钮，回到数据选择窗口。此数据集具有 1000 个样本数据，输入为 2×1000 矩阵，输出划分为 4 类。

③ 单击“Next”按钮，进入到网络尺寸设置窗口，如图 7-48 所示。

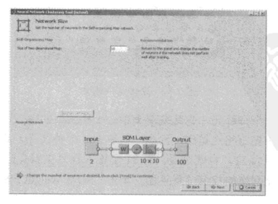


图 7-48 nctool 网络尺寸设置窗口

此二维自组织映射网络的尺寸默认设置为 10，意味着每一边有 10 个节点，神经元总数为 100 个，我们选择此默认值作为待建立网络的设置。

④ 单击“Next”按钮开始建立网络，此时会进入到网络训练窗口，单击“Train”按钮即开始网络训练，训练过程中弹出 nntraintool 窗口。此过程与 nprtool 基本一致。

训练设置的最大迭代次数为 200，训练完成后，回到网络训练窗口。此时窗口如图 7-49 所示。

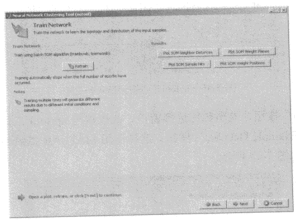


图 7-49　训练完毕后的网络训练窗口

在此窗口中单击“Plot Sample Hits”即可查看此 SOM 网络对数据进行聚类的命中结果，如图 7-50 所示。

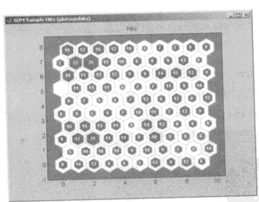


图 7-50　聚类命中结果

图中显示了与每一个神经元（也就是聚类中心）相关联的输入训练数据的向量数目。该网络为 10×10 的二维拓扑格点结构，因此一共有 100 个神经元。从图中可见，具有最大命中数的神经元命中数为 31，也就是说，有 31 个输入样本被聚类到了此神经元中心处。

同样，可以查看此网络的权值位置平面，单击“SOM Weight Planes”按钮，绘出图形如图 7-51 所示。

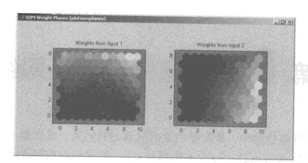


图 7-51 权值位置平面

此图显示了输入向量的每一个元素所对应的权值分布情况，对于此例，输入向量为 2 维，因此有两个输入。其中黑色的部分代表大的权值，浅色的部分代表小的权值。

如果两个输入对应的权值分布平面非常相似，那么说明这两个输入相关度是非常高的。在此例下，两者的权值分布平面相差很大，说明这两个输入的相关性非常低。

⑤ 单击“Next”按钮，进入网络评估窗口。再次单击“Next”按钮，即进入数据保存窗口。

数据保存完毕之后，单击“Finish”按钮，即完成此次训练。此过程同应用 nprtool 工具时一样，因此不再详细叙述。

7.5 小结

本章通过实例操作，详细讲述了 MATLAB 神经网络的 GUI 工具，包括基础 GUI 工具 nntool、函数拟合 GUI 工具 nftool、模式识别 GUI 工具 nprtool，以及数据聚类 GUI 工具 nctool。

熟练应用这些 GUI 工具，才能够快速上手，高效开展神经网络具体应用的工作。



第 8 章　感知器神经网络

在众多的神经网络结构中，最简单的是感知器神经网络。感知器（Perceptron）神经网络传递函数采用的是阈值函数（硬限值传递函数），输出只具有两个状态，它是美国学者 Rosenblatt 等人于 1957 年在 M-P 模型和 Hebb 学习规则的基础上提出来的，可以说是最早的神经网络模型。

通过对网络权值和偏差进行训练，可以使感知器在一组输入向量下获得 0 或 1 的目标响应，正是由于这个特点，感知器神经网络特别适合于简单的模式分类问题，但是通常也只能用来实现线性可分的两类模式识别。

8.1 感知器神经网络结构

感知器虽然是最简单的模拟生物神经网络的人工模型，但因为感知器模型包含了自组织、自学习的思想和概念，直观而易于理解，因此在神经网络研究中具有重要的意义。

8.1.1 感知器神经元模型

图 8-1 给出了一个感知器神经元的模型，该神经元具有 R 个输入变量 p_i，赋以不同的权值 $w_{1,i}$ 之后相加，并加入偏差 b（b 通常也称为神经网络的阈值，本书中称为偏差），其计算公式表示为：

$$n = \sum_{i=1}^{R} p_i w_{1,i} + b$$

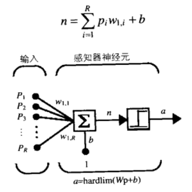


图 8-1　感知器神经元模型示意图

感知器神经元采用阈值函数（hardlim）作为网络传递函数，如图 8-2 所示，该函数根据输入是否大于 0，输出为 1 或者 0。

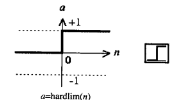


图 8-2 阈值传递函数图

正是基于阈值函数的二值特性，感知器神经元可以将输入的 R 维空间依靠平面或直线线性地划分成两个互不重叠的区域。当输入 $n<0$ 时，输出为 0；当输入 $n>0$ 时，输出为 1。

对于一个具有二维输入的感知器神经元，比如其权值和偏差分别为：

$$w_{1,1}=-1\text{，}\quad w_{1,2}=1\text{，}\quad b=1$$

它对输入 p_1, p_2 构成的二维平面的划分情况如图 8-3 所示。二维平面被决策边界线分割为两个区域，此输入空间所对应的决策边界线满足的方程就是 $\mathbf{Wp}+\boldsymbol{b}=0$，决策边界线与权值矢量 $\boldsymbol{W}$ 正交。在直线的上部，输入向量对应感知器神经元的输出等于 1，而直线下部，输入向量对应感知器神经元的输出等于 0。

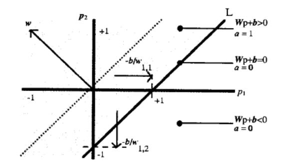


图 8-3 感知器神经元对二维平面的分割

从图 8-3 中可以看到，通过改变偏差 $\boldsymbol{b}$，可以将决策边界线上下移动；而调整权值矢量 $\boldsymbol{W}$，则可以改变决策边界线的斜率，实现对输入空间的不同线性划分。而调整偏差和权值的过程，也就是对网络进行训练的过程。

8.1.2 单层感知器神经网络结构

一个具有 R 维输入，S 维输出的单层感知器神经网络结构如图 8-4 所示。此神经网络结构可以表示为两种形式，其中左图 $w_{i,j}$ 代表第 j 个输入对第 i 个神经元的权值；而右图则以输入向量、权值矩阵的形式表示。

下面以图 8-4 右图的形式来对网络结构进行说明，输入向量 $\boldsymbol{P}$ 为 $R\times1$ 维，权值矩阵 $\boldsymbol{W}$ 为 $S\times R$ 维，偏差矢量 $\boldsymbol{b}$ 为 $S\times1$ 维，最终输出向量 $\boldsymbol{a}$ 为 $S\times1$ 维，从而感知器的输出以向量的形式可以表示为：

$$a = \text{hard lim}(\mathbf{W}\mathbf{p} + \boldsymbol{b})$$

其中，根据 hard lim 函数的运算结果，输出向量 $\boldsymbol{a}$ 中的各个元素 a_i 取值范围都为{0, 1}二值区间。

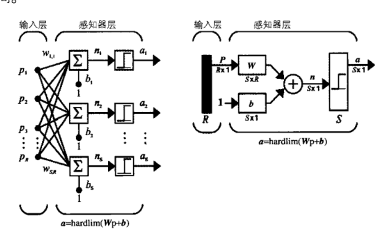


图 8-4　单层感知器神经网络结构示意图

在后面的讨论中我们将会看到，由于感知器训练算法只能训练单层的网络结构，因此感知器神经网络通常也采用单层结构，这在一定程度上限制了能够应用感知器神经网络的范围。

8.2 感知器学习规则

感知器神经网络的可调参数包括网络权值和偏差，这两者的调整是通过一定的学习规则来实现的，通过对一定输入样本的训练使得调整后的权值与偏差对输入向量产生期望的目标响应，使得经过训练的网络能够完成对某些特定的信息进行处理的任务。

通常，根据学习过程中是否存在期望目标响应，神经网络的学习算法可分为有监督学习（supervised learning）和无监督学习（unsupervised learning）两种，

在有监督学习的情况下，需要对网络提供一系列正确的输入/输出训练样本：

$$\{\boldsymbol{p}_1,\boldsymbol{t}_1\},\{\boldsymbol{p}_2,\boldsymbol{t}_2\},\ldots,\{\boldsymbol{p}_Q,\boldsymbol{t}_Q\}$$

其中 $\boldsymbol{p}_i$ 是输入样本，$\boldsymbol{t}_i$ 是期望的目标响应。将期望的目标响应与训练时产生的目标输出进行比较，获得网络误差，然后应用学习规则对网络权值和偏差进行调整，使得网络响应更接近于正确的目标响应。

感知器神经网络的学习规则属于有监督学习一类的，训练的目的是得到一个对特定的输入能够产生正确响应的网络。

对于无监督学习的情况，训练中不提供目标响应样本，网络权值与偏差仅仅根据输入来进行调整。大多数的这种学习规则采用聚类法，将输入归于有限的种类，无监督学习规

则通常只适用于特定的网络。

8.2.1 感知器网络学习算法

通过训练，感知器网络可以对输入向量产生期望的 0 或 1 的目标输出，下面对其学习规则进行介绍。

假设 $\boldsymbol{p}$ 是训练时提供的输入向量，$\boldsymbol{t}$ 是期望的输出向量，训练时感知器实际的输出向量为 $\boldsymbol{a}$，训练的目标是尽量地减小偏差 $\boldsymbol{e}=\boldsymbol{t}-\boldsymbol{a}$。要注意，提供的期望输出向量 $\boldsymbol{t}$ 的取值范围必须为 0 和 1，因为感知器神经网络只能够输出这两个值。根据输入向量和网络输出的值，感知器的学习规则是根据下面几种情况来进行参数调整的。

（1）对于提供的输入向量，如果对第 i 个神经元的输出响应是正确的，即 $e_i = t_i - a_i = 0$，则不对这个神经元的权值矢量 $\boldsymbol{w}_i$ 进行修改。

（2）如果某个神经元的输出为 0，而期望响应为 1，则将此时权值矢量 $\boldsymbol{w}_i$ 加上输入向量 $\boldsymbol{p}$ 作为新的权值矢量。这样可以使得神经元权值矢量更加接近输入向量，增加将输入向量划归为 1 的概率。

（3）如果某个神经元的输出为 1，而期望响应为 0，则将此时权值矢量 $\boldsymbol{w}_i$ 减去输入向量 $\boldsymbol{p}$ 作为新的权值矢量。这样可以使得神经元权值矢量更加远离输入向量，从而增加将输入向量归类于 0 的概率。

以上三种情况的权值修正公式可以由一个表达式给出，即：

$$\Delta \boldsymbol{w}_i = (t_i - a_i)\boldsymbol{p} = e_i \boldsymbol{p}\text{，其中，} i = 1,2,\ldots,S$$

而偏差的修正公式则为：

$$\Delta b_i = t_i - a_i = e_i\text{，其中，} i = 1,2,\ldots,S$$

对于单层感知器神经网络整体而言，其权值矩阵与偏差矢量的修正就可以写为：

$$\Delta \boldsymbol{W} = (\boldsymbol{t} - \boldsymbol{a})\boldsymbol{p}^{\mathrm{T}} = \boldsymbol{e}\boldsymbol{p}^{\mathrm{T}}$$

$$\Delta \boldsymbol{b} = (\boldsymbol{t} - \boldsymbol{a}) = \boldsymbol{e}$$

从而新的权值和阈值为：

$$\boldsymbol{W}_{\text{new}} = \boldsymbol{W} + \Delta \boldsymbol{W} = \boldsymbol{W} + \boldsymbol{e}\boldsymbol{p}^{\mathrm{T}}$$

$$\boldsymbol{b}_{\text{new}} = \boldsymbol{b} + \Delta \boldsymbol{b} = \boldsymbol{b} + \boldsymbol{e}$$

其中：$\boldsymbol{e} = \boldsymbol{t} - \boldsymbol{a}$。

感知器学习规则的本质就是权值变化量等于正负输入向量。每进行一次权值修正，感知器产生正确响应的几率就会有所提高，此学习规则属于梯度下降算法。对于此算法已经证明，如果解存在，则该算法在经过有限次的迭代运算后能够收敛到正确的目标向量。

MATLAB 中用来计算感知器权值修正量的函数为 learnp，我们将在 8.3 节中进行介绍。

8.2.2 标准化感知器网络学习算法

我们知道，一般的感知器学习规则权值调整采用的公式是：

$$\Delta W = (t-a)p^{\mathrm{T}} = ep^{\mathrm{T}}$$

可以见到，输入向量 p 越大，则权值的变化也就越大，因此当输入样本中存在奇异样本的时候，会让权值产生一个非常大的变化，而占大多数的小样本则需要花费很多的时间才能够抵消少量的奇异样本引起的权值的变动。

因此对于输入样本中含有奇异样本的情形，感知器网络的训练时间会大大增加，为了解决这个问题，人们提出了一种改进的感知器学习规则，即所谓的标准化感知器学习规则。

标准化的学习规则的目标是使得奇异样本和其他正常样本对权值产生均衡的影响，所采取的权值修正公式为：

$$\Delta W = (t-a)\frac{p^{\mathrm{T}}}{\|p\|} = e\frac{p^{\mathrm{T}}}{\|p\|}$$

这样相对于原始的学习规则而言，标准化学习规则所用的时间稍长一些，但是通过归一化的方法，避免了权值调整过程中发生少数大样本抵消大量小样本的作用的情况，从而可以应用于奇异样本的输入情形。

在神经网络工具箱中，标准化感知器网络学习算法是通过函数 learnpn 实现的，我们将在后面的章节中进行介绍。

8.3 感知器网络的 MATLAB 实现

MATLAB 工具箱为我们提供了关于感知器神经网络操作的丰富函数，本节我们将针对网络生成、网络仿真、网络学习与训练，以实例的形式对与感知器网络相关的 MATLAB 工具箱函数进行介绍。

8.3.1 感知器网络的生成

利用 MATLAB 工具箱函数 newp 可以生成一个感知器神经网络，其调用格式如下：

net = newp(PR, *S*, TF, LF)

其中各参数意义如下：

- PR 为 $R\times 2$ 阶矩阵，由 R 个输入元素的最小值与最大值组成；
- S 为神经元的数目；
- TF 为传递函数，默认值为 'hardlim'；
- LF 为学习函数，默认值为 'learnp'。

函数返回值 net 是一个结构体，代表生成的感知器神经网络的对象。因为 TF，LF 具

有默认的'hardlim'与'learnp'值，因此调用该函数时，生成一般的感知器时可以采用下述默认调用格式：

$$net = newp(PR, S)$$

【例 8-1】 感知器网络生成实例。生成一个一维输入、具有一个神经元的感知器神经网络，输入的最小值和最大值为[0,2]，输出值为 0 或者 1。

解：题目要求的神经网络可以应用如下命令来产生：

```
P = [0 2];
T = [0 1];
net = newp(P,T);
```

以上命令将生成一个新的名为 net 的感知器神经网络对象结构，关于网络对象结构的内容在上一节中已有介绍，在“自定义神经网络”一章中还将详细讲解，此处不再赘述。

我们可以输入命令来查看生成的感知器神经网络的内部变量和参数。例如，网络的输入权值保存在结构体 net.intputweights 中，可以利用下列语句来查看：

```
inputweights = net.inputweights{1,1}
```

输出结果为：

```
inputweights =
       delays: 0
      initFcn: 'initzero'
        learn: 1
     learnFcn: 'learnp'
   learnParam: []
         size: [1 1]
     userdata: [1x1 struct]
    weightFcn: 'dotprod'
  weightParam: [1x1 struct]
```

可以看到的信息是，网络采用了默认的传递函数 hardlim 和学习算法 learnp。而网络传输给阈值函数 hardlim 的数值是权值矩阵与输入向量的点积（dotprod）。网络权值初始化函数采用了 initzero，这表示对权值初值采取了赋 0 处理。

同样，我们也可以查看网络偏差的信息，在 MATLAB 命令窗口中输入：

```
biases = net.biases{1}
```

输出结果为：

```
biases =
      initFcn: 'initzero'
        learn: 1
     learnFcn: 'learnp'
   learnParam: []
         size: 1
     userdata: [1x1 struct]
```

这说明偏差初值也赋了 0 值。

8.3.2 感知器网络的仿真

在利用 newp 函数生成了感知器网络之后，我们可以利用 sim 函数来对网络进行仿真，得到在一定输入向量下的网络响应。

【例 8-2】 感知器网络仿真实例。生成一个输入向量取值范围为[-2,2;-2,2]、输出响应取值范围为[0,1]的单神经元的感知器网络，利用 sim 函数进行仿真。

解：题目要求的神经网络可以利用下列语句生成：

```
net = newp([-2 2;-2 2],[0 1]);
```

这样生成的网络其权值与偏差均为 0。如果我们想要改变其权值与偏差，可以通过下列的语句进行赋值，在 MATLAB 命令窗口中输入：

```
net.IW{1,1}= [-1 1];
net.b{1} = [1];
```

这样就将权值赋值为[-1,1]，而偏差 b 赋值为 1。更改后可以通过命令进行查看，在 MATLAB 命令窗口中输入：

```
net.IW{1,1}
```

得到：

```
ans =    -1     1
```

输入命令：

```
net.b{1}
```

输出结果为：

```
ans =     1
```

下面通过 sim 函数来对网络进行仿真，并查看此网络对于特定输入向量的响应输出。查看第一个输入向量，输入下列命令：

```
p1 = [1;1];
a1 = sim(net,p1)
```

得到：

```
a1 =     1
```

查看第二个输入向量，输入下列命令：

```
p2 = [1;-1];
a2 = sim(net,p2)
```

得到：

```
a2 =     0
```

可以看到，网络对两个不同的输入向量产生的响应分别为 1，0。我们也可以将这两个

输入向量组成一个输入向量序列，则网络响应输出也是一个向量序列，MATLAB 命令及执行情况如下：

```
p3 = {[1;1] [1;-1]};
a3 = sim(net,p3)
```

得到：

```
a3 =     [1]    [0]
```

这样我们就实现了对感知器网络的仿真。

8.3.3 感知器网络的初始化

在上一节中，我们看到，在默认情况下应用 newp 函数生成的感知器网络的权值与偏差都初始化为 0，而在仿真过程中则改变了这两者的值，如果我们需要重置感知器权值和偏差的话，可以使用 init 函数。

【例 8-3】 感知器网络初始化实例。利用【例 8-2】中的感知器网络，练习感知器网络的初始化。

解：利用下面的语句生成感知器网络：

```
net = newp([-2 2;-2 2],[0 1]);
```

分别对其权值偏差进行查看，输入命令：

```
wts = net.IW{1,1}
```

得到：

```
wts =     0     0
```

输入命令：

```
bias = net.b{1}
```

输出结果为：

```
bias =     0
```

下面我们设置网络权值为[3,4]，设置偏差为 5，输入命令：

```
net.IW{1,1} = [3,4];
net.b{1} = 5;
```

重新检查改变后的权值与偏差，得到：

```
wts =     3     4
bias =     5
```

现在，利用 init 函数对网络参数进行重置，输入命令：

```
net = init(net);
```

再度查看权值与偏差，即有：



```
wts =     0     0
bias =     0
```

至此，我们完成了网络参数的重置。

除了初始化为 0 值之外，也可以利用 Init 函数将感知器网络参数初始化为其他的值，例如在上例中继续输入下面的语句，可以将感知器的初始权值与偏差重新初始化为随机数：

```
net.inputweights{1,1}.initFcn = 'rands';
net.biases{1}.initFcn = 'rands';
net = init(net);
```

查看权值与偏差，输出结果为：

```
wts =    0.8116   -0.7460
biases =    0.6294
```

由此可见，感知器的权值与偏差都初始化为了随机数。

8.3.4 感知器网络的学习和训练

1962 年，Rosenblatt 宣布，神经网络可以学会它能够表示的任何东西，这极大地促进了神经网络的发展，对于感知器神经网络，我们在上一节已经介绍过其学习算法，这一节将对 MATLAB 中相关的函数进行介绍。

给定一定的输入向量 $\boldsymbol{p}$，以及偏差 $\boldsymbol{e}$，在 MATLAB 中应用感知器训练函数 learnp 可以计算需要对网络权值 $\boldsymbol{W}$ 和偏差 $\boldsymbol{b}$ 进行的修正，此过程采用的算法为原始的感知器网络学习算法。

learnp 函数的完整调用格式为：

$$[dW, LS] = learnp(\boldsymbol{W}, \boldsymbol{P}, Z, N, A, T, \boldsymbol{E}, gW, gA, D, LP, LS)$$

此函数涉及的参数比较多，在感知器的学习中，通常只需要用到其中 $\boldsymbol{W}$、$\boldsymbol{P}$、$\boldsymbol{E}$ 三个参数，它们分别为网络原始权值、输入向量和误差向量，其余的参数置空即可。dW 为返回的权值调整量。一般情况下应用感知器时，采用简化的调用格式即可，下面以示例说明其用法。

【例 8-4】 感知器网络学习实例。以【例 8-3】中的双输入的单神经元感知器网络为例，练习感知器网络学习函数 learnp 的使用。

解： ① 首先生成网络。输入命令：

```
net = newp([-2 2;-2 2],[0 1]);
```

然后将偏差置为 0，权值置为 1 和 0.8，输入如下命令：

```
net.b{1} = [0];
w = [1 -0.8];
net.IW{1,1} = w;
```

给定的输入、输出目标向量分别为：

```
p = [1; 2];
t = [1];
```

可以通过 sim 函数计算得到网络相对于期望响应的误差，输入命令：

```
a = sim(net,p)
```

输出结果为：

```
a =     0
```

输入命令：

```
e = t-a
```

输出结果为：

```
e =     1
```

② 利用 learnp 函数对网络权值进行调整。输入命令：

```
dw = learnp(w,p,[],[],[],[],e,[],[],[])
dw =     1     2
```

新的权重可以通过下面的语句计算得到：

```
w = w + dw
```

输出结果为：

```
w =    2.0000    1.2000
```

已经证明，通过循环调用 learnp 函数，可以保证经过有限次运算之后，神经网络的输出会收敛到期望的目标响应。

在给定的输入向量下，通过不断地调用 sim 和 learnp 函数，根据输出误差调整网络权值与偏差，最终将会得到针对此输入期望响应的最优权值和偏差，给出解决此问题的最优感知器网络，也就完成了一次对于网络的训练过程。

通过反复调用函数 adapt，或者直接调用 train 函数都可以完成这个过程。

（1）adapt 函数

调用 adapt 函数对感知器网络进行训练的格式为：

$$[\mathrm{net}, \boldsymbol{a}, \boldsymbol{e}] = \mathrm{adapt}(\mathrm{net}, \boldsymbol{p}, \boldsymbol{t})$$

其中等式右边的 net 为训练前的网络对象，左边的 net 为训练后的网络对象，$\boldsymbol{a}$ 和 $\boldsymbol{e}$ 分别为训练后的网络输出和误差。

（2）train 函数

train 函数的调用格式如下：

$$\mathrm{net} = \mathrm{train}(\mathrm{net}, \boldsymbol{p}, \boldsymbol{t})$$

其中 net 是网络对象，$\boldsymbol{p}$ 为输入向量，$\boldsymbol{t}$ 为期望输出响应。

需要注意的是，函数 train 并不能够保证网络完全产生期望的目标响应。在完成一次训练之后，仍然需要对新的权值和偏差下的网络进行检验，以确保对于每一个输入向量都能够产生期望的输出响应。如果发现新的网络并没有产生所期望的响应的话，就需要重复调用 train 函数对网络进行进一步的训练。

如果经过多次训练，网络性能仍然达不到要求的话，那么就需要仔细考虑一下，此问题是否可以通过感知器神经网络来解决，感知器网络的局限性将在下一节进行介绍。

下面通过一个简单的问题来介绍 train 函数的使用。

【例 8-5】 感知器网络训练函数 train 使用实例。考虑一个单个神经元构成的具有两维输入向量的感知器神经网络，假定用于训练的输入向量和期望输出如下：

$$\{\boldsymbol{p}_1=\begin{bmatrix}2\\2\end{bmatrix},t_1=0\}，\{\boldsymbol{p}_2=\begin{bmatrix}1\\-2\end{bmatrix},t_2=1\}，\{\boldsymbol{p}_3=\begin{bmatrix}-2\\2\end{bmatrix},t_3=0\}，\{\boldsymbol{p}_4=\begin{bmatrix}-1\\1\end{bmatrix},t_4=1\}$$

对网络进行训练，使之完成此分类问题。

解： ① 生成感知器神经网络。

仍然利用 newp 生成一个感知器神经网络，初始的网络权值和偏差均为 0，输入命令：

```
net = newp([-2 2;-2 2],[0 1]);
```

首先利用第一个输入向量来对网络进行训练，设置迭代次数为 1，则 train 函数只对网络参数进行一次调整。输入命令：

```
net.trainParam.epochs = 1;
net = train(net,p,t);
```

关闭弹出的图形窗口，回到 MATLAB 命令工作区，查看新的权值和偏差，输入命令：

```
w = net.iw{1,1}, b = net.b{1}
```

得到：

```
w =    -2    -2
b =    -1
```

这样，经过训练后的网络权值变成了[-2,-2]，偏差变成了-1，这里仅仅采用了 $\boldsymbol{p}_1$ 作为输入向量来进行训练，接下来，还需要应用 $\boldsymbol{p}_2$，$\boldsymbol{p}_3$，$\boldsymbol{p}_4$ 三个输入向量对网络进行训练，在此基础上进一步调整权值和偏差，从而得到最终符合这四个期望输出响应的网络。

实际上，这个过程可以通过 train 函数一次完成，操作过程如下所示。在 MATLAB 命令窗口中输入：

```
net = newp([-2 2;-2 2],[0 1]);
net.trainParam.epochs = 1;
```

输入命令，定义输入向量和期望输出序列：

```
p = [[2;2] [1;-2] [-2;2] [-1;1]]
t = [0 1 0 1]
```

输入命令，应用此输入输出序列对网络进行训练：

```
net = train(net,p,t)
```

输入命令，查看训练得到的权值和偏差：

```
w =net.iw{1,1}, b = net.b{1}
```

输出结果为：

```
w =    -3    -1
b =     0
```

② 训练网络。用 sim 函数对训练生成的网络进行仿真，检验对于每一个输入向量是否能够得到期望的输出响应。

输入命令：

```
a = sim(net,p)
```

输出结果为：

```
a =       0    0    1    1
```

此时会发现，生成的网络并没有符合我们的预期，我们需要对网络进行多次迭代调整，所以可以改变最大迭代值，重新训练。输入命令：

```
net.trainParam.epochs = 1000;
net = train(net,p,t);
```

此时弹出的图形窗口中显示“performance goal meet”。回到 MATLAB 命令工作区，再查看权值与误差，输入命令：

```
w =w = net.iw{1,1}, b = net.b{1}
```

输出结果为：

```
w =    -2    -3
b =     1
```

再次进行仿真，输入命令：

```
a = sim(net,p)
```

输出结果为：

```
a =         0          1          0          1
```

输入命令：

```
error = a-t
```

输出结果为：

```
error =         0          0          0          0
```

至此，通过训练，感知器神经网络收敛到了期望的输出响应。train 函数除了用于感知

器神经网络的训练之外，也可以进行其他种类的神经网络的训练，在后面的章节中会继续介绍。

8.4 感知器网络的局限性

由于感知器的出现，神经网络在 20 世纪 40 年代就初步展现了它的功能和诱人的发展前景，20 世纪 60 年代，在 Rosenblatt 等人的推进下，感知器的研究又获得了更大的发展。正当人们为取得的进展高兴的时候，却发现了很多应用神经网络无法解决的问题。

Minsky 进行研究后得出结论，单层的神经网络无法解决异或等基本的问题，这使得神经网络的发展一下受到了阻碍，从第一个高潮期进入了反思期。在本节我们对感知器网络所具有的局限性进行介绍。

8.4.1 单层感知器网络的局限性

由于感知器神经网络在结构和学习规则上的局限性，使得其应用也受到一定的限制。它的局限性主要包括以下几点：

（1）由于采用 harlim 也就是阈值函数作为网络传递函数，所以感知器的输出只能为 0 或 1 两个值。

（2）单层感知器神经网络只能够解决线性可分的模式分类问题，也就是说，只能够对线性可分的输入样本进行分类识别，而对于线性不可分的输入样本就显得无能为力了，即使是通过网络学习，也无法达到分界点。不过，对于线性可分的情形，只要通过训练，总能在有限的循环次数内使网络达到期望值，这是经过证明的。

（3）Minsky 证明，单层感知器网络无法实现简单的异或逻辑。

（4）采用常规感知器学习算法的感知器神经网络，如果遇到含有奇异样本的输入向量的时候，即少数样本相对于其他样本非常大时，对其训练所需要收敛时间将会变得很长。这一点已经在前面的标准化感知器学习算法中介绍了。

8.4.2 多层感知器神经网络

为改善单层感知器网络对于较为复杂模式分类问题的局限性，20 世纪 60 年代人们发展了多层感知器神经网络，在网络中包含了一层以上的感知器神经元，这样在输入层与输出层之间增加一些隐层，网络得以实现一些更复杂的功能，一个多层感知器模型如图 8-5 所示。

多层感知器可以用于解决一些单层感知器所无法解决的比较困难的问题。例如我们要将 4 个输入向量分为 4 类，应用单层感知器神经网络只能得到一条决策边界线，无法进行 4 类模式的划分，然而应用多层神经元构成的感知器网络，我们就可以通过两条线将输入向量划分为 4 类，从而解决这一类问题。

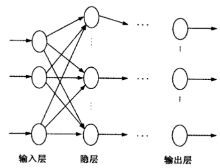


图 8-5 多层感知器模型示意图

此外，对于 Minsky 提出的单层感知器所无法解决的异或问题，应用二层感知器网络也可以成功地解决。

8.5 感知器神经网络设计实例

下面通过 4 个综合实例，讲述感知器神经网络的设计。

8.5.1 输入向量的二类划分

【例 8-6】 利用一个二输入感知器网络，完成对 4 个二维输入向量的分类。下面的二维数组 ***P*** 每一列对应一个二维输入向量，***T*** 是对应向量期望的目标响应。

解： ① 在 MATLAB 命令区进行 ***P***，***T*** 的定义，并用函数 plotpv 在二维输入平面上画出输入向量与期望响应示意图，如图 8-6 所示。输入命令：

```
P = [ -0.5 -0.5 +0.3 -0.1;
    -0.5 +0.5 -0.5 +1.0];
T = [1 1 0 0];
plotpv(P,T);
```

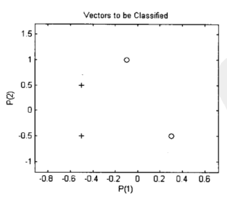


图 8-6 输入向量与期望响应样本图

② 生成网络。我们构造的神经网络必须能够将这 4 个输入向量 ***P*** 依据期望响应 ***T*** 进行正确的二类划分。所幸的是，对于这类简单的二类模式划分问题，感知器神经网络是足以胜任的。

首先，利用 newp 函数生成一个感知器神经网络对象，输入向量的最小和最大值分别为-1 和 1，表示为[-1 1;-1 1]，层数为 1。输入命令：

```
net = newp([-1 1;-1 1],1);
```

我们可以在输入向量与期望响应图加上生成感知器网络的决策分界线，这样就可以看到此网络对于输入向量的划分情况。

然而到目前为止，感知器网络的权值与偏差都是初始化为 0 的，因此在画出的图上仍然看不到决策分界线。如果分界线存在的话，可以使用以下命令绘出分界线：

```
plotpv(P,T);
plotpc(net.IW{1},net.b{1});
```

③ 训练网络。由于初始化的网络不能完成对输入向量的划分，因此我们需要对网络进行训练，使其能够适用于此问题的解决。

这里，我们将介绍另一个训练函数 adapt。此函数执行的结果将会返回一个新的网络对象、输出，以及误差。输入命令：

```
net.adaptParam.passes = 3;
net = adapt(net,P,T);
plotpc(net.IW{1},net.b{1});
```

执行上面的命令，我们就可以看到结果，如图 8-7 所示，训练后的感知器的网络决策分界线成功地对两类输入样本进行了分类。

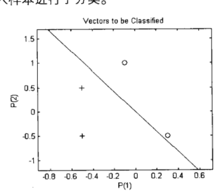


图 8-7 训练后的网络对输入样本的分类

④ 网络仿真。现在，可以应用 sim 函数来进行网络仿真，实现对任何其他输入向量的模式划分了。

用 MATLAB 命令完成此过程，并将计算结果添加在图 8-7 上，最后得到的结果如图 8-8 所示。可以看到，新的输入向量在箭头标记的位置处，根据响应 ***a*** 的值被划归到了上

半平面。

输入的 MATLAB 命令为：

```
hold on;
p = [0.7; 1.2];
a = sim(net,p);
plotpv(p,a);xlim([-1,0.8]),ylim([-1,1.5]);
```

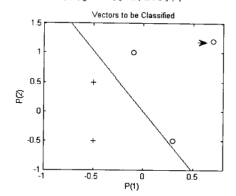


图 8-8 新输入样本的分类仿真结果

在此实例中，我们首先利用 newp 生成感知器网络，利用 adapt 函数对网络进行训练，然后通过 sim 进行网络仿真，从而应用感知器网络完成了一个简单的二类模式划分问题。简单的二类模式划分是感知器最适用的场合。

8.5.2 奇异样本输入向量的训练

【例 8-7】 应用一个含有奇异样本的输入向量与期望样本序列，对一个感知器神经网络进行训练，使其达到二类模式划分的目标。

在此例中我们将会看到，由于输入序列中存在一个非常大的奇异样本，训练时间将会远远大于正常样本所需要的时间。

解： ① 定义输入向量与期望响应样本，然后利用 plotpv 函数画出输入向量与期望响应样本。输入命令：

```
P = [-0.5 -0.5 +0.3 -0.1 -40;
-0.5 +0.5 -0.5 +1.0 50];
T = [1 1 0 0 1];
plotpv(P,T);
```

绘出图形如图 8-9 所示。

从图中可以看到，在 5 个输入样本点中，有 4 个在数值上是非常小的，但是第 5 个样本处于图片的左上角，数值上远远大于 4 个正常值。而构造的感知器网络需要根据期望响应的值将这个 5 个点线性地划分为两类。

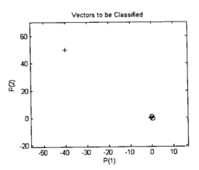


图 8-9　输入向量与期望响应样本图

(2) 生成网络。首先利用 newp 函数构造神经网络。为了包含所有的输入样本点，设定输入样本变化范围为[-40 1;-1 50]，生成网络为 1 层。生成网络之后，在图上画出其决策分界线。输入如下 MATLAB 命令：

```
net = newp([-40 1;-1 50],1);
hold on
linehandle = plotpc(net.IW{1},net.b{1});
```

初始生成网络的权值与偏差都为 0，分界线在图上无法看出，因此无法进行正确的分类，需要对其进行训练。

(3) 训练网络。利用 adapt 函数对网络进行训练，此处将每次 adapt 函数训练过程迭代次数定为 3，持续到误差为 0 为止，这是通过在 while 循环条件内利用函数 sse 计算误差向量的平方和来进行控制的。

训练完成后，将训练后的网络决策分界线绘于输入/输出向量图上，如图 8-10 所示。输入命令：

```
E = 1;
net.adaptParam.passes = 3;
while (sse(E))
   [net,Y,E] = adapt(net,P,T);
   linehandle = plotpc(net.IW{1},net.b{1},linehandle);
   drawnow;
end
```

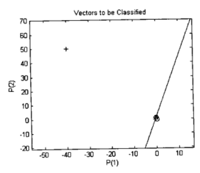


图 8-10　训练后的网络对输入样本的分类

由于输入样本中存在一个非常大的奇异样本，这大大地增加了网络训练的时间。不过，尽管花费了更长的训练时间，感知器网络还是通过学习对输入样本实现了正确的分类。

④ 网络仿真。现在我们可以利用 sim 函数来进行网络仿真，通过一个新的输入向量[0.7; 1.2]来检验生成网络的性能。

通过 hold on 命令，保持原有图像，在其上增加新的绘图，在新增绘图完成后，通过 hold off 释放图像资源。输入命令：

```
p = [0.7; 1.2];
a = sim(net,p);
plotpv(p,a);
hold on;
plotpv(P,T);
plotpc(net.IW{1},net.b{1});
hold off;
```

在图上显示出对新输入样本的划分情况，如图 8-11 所示。

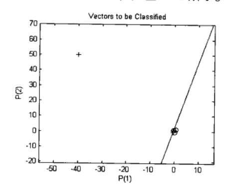


图 8-11　新输入样本的分类仿真结果

由于样本点太小，我们可以缩放图像，便于查看。输入命令：

```
axis([-2 2 -2 2]);
```

得到放大后的图像，如图 8-12 所示。在图中新的输入样本点由箭头标出。

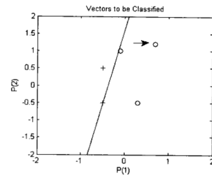


图 8-12　放大后的分类结果

从本实例中看到，如果输入样本中存在非常大的奇异样本，在花费更长的训练时间的情况下，感知器网络仍然成功地实现了对输入向量的线性分类。对于这种情况如何有效地减少训练时间，我们将在下一节中通过实例进行介绍。

8.5.3 标准化感知器学习规则实例

【例 8-8】 同样对【例 8-7】中包含奇异样本的二维输入向量进行分类。此例中不采用原始的感知器学习规则，而是采用标准化感知器学习规则，应用 learnpn 函数对网络进行训练。

解：① 通过下列语句定义输入向量与期望响应，并作图。输入命令：

```
P = [ -0.5 -0.5 +0.3 -0.1 -40;
     -0.5 +0.5 -0.5 +1.0 50];
T = [1 1 0 0 1];
plotpv(P,T);
```

② 生成网络。仍然利用 newp 函数生成网络，但这回我们不采用默认的学习规则，而是在 newp 函数中指定学习规则为 learnpn，即标准化感知器学习规则。输入命令：

```
net = newp([-40 1;-1 50],1,'hardlim','learnpn');
hold on
linehandle = plotpc(net.IW{1},net.b{1});
```

③ 训练网络。对网络进行训练，并绘出训练后的网络的决策分界线。输入命令：

```
E = 1;
net.adaptParam.passes = 3;
while (sse(E))
   [net,Y,E] = adapt(net,P,T);
   linehandle = plotpc(net.IW{1},net.b{1},linehandle);
   drawnow;
end
```

绘出图形如图 8-13 所示。

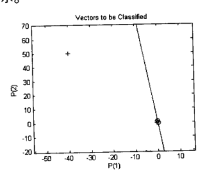


图 8-13 训练后网络对输入样本的分类

输入命令 axis 调整坐标轴显示范围，对图像进行放大显示，从而可以更清楚地看到其

分类情况。

```
axis([-2 2 -2 2]);
```

绘出图形如图 8-14 所示。

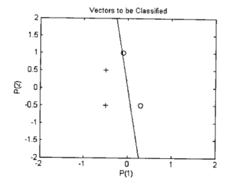


图 8-14 放大显示的样本分类结果

可以看到，相对于利用原始学习规则训练得到的感知器网络图 8-10，图 8-13 中网络的决策分界线发生了变化。而训练过程的执行时间也变得更短，这是因为，标准化感知器学习算法更加适用于输入样本变动非常大的场合。

8.5.4 线性不可分样本问题

感知器对于线性可分样本的二类划分是非常适用的，然而对于线性不可分的输入向量，感知器就无能为力了。这一节我们来看这样的一个实例，这有助于加深我们对感知器神经网络的局限性的理解。

【例 8-9】 假定二维输入向量与期望响应为 ***P*** = [-0.5 -0.5 +0.3 -0.1 -0.8; -0.5 +0.5 -0.5 +1.0 +0.0]；***T*** = [1 1 0 0 0]，通过一个二输入的感知器神经网络对其进行分类。

解： ① 定义 ***P***、***T*** 后，利用 plotpv 函数对输入向量进行图形显示。输入命令：

```
P = [ -0.5 -0.5 +0.3 -0.1 -0.8;
     -0.5 +0.5 -0.5 +1.0 +0.0 ];
T = [1 1 0 0 0];
plotpv(P,T);
```

绘出图形如图 8-15 所示。

② 生成网络。对于上面的输入样本，我们尝试用感知器神经网络来对其进行划分。首先利用 newp 函数生成一个单层感知器网络，并作图。输入命令：

```
net = newp([-40 1;-1 50],1);
hold on
plotpv(P,T);
linehandle=plotpc(net.IW{1},net.b{1});
```

由于生成的网络权值与偏差都初始化为 0，因此无法进行分类，图上也无法显示决策

分界线。

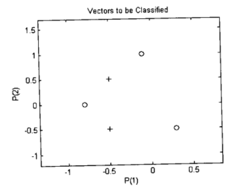


图 8-15　输入向量与期望响应样本图

③ 训练网络。我们希望能够通过对网络进行训练，使得网络能够对输入样本进行分类。因此利用函数 adapt 进行训练，每次 adapt 函数执行过程设置 3 次迭代，循环进行 25 次，然后在图上画出训练后网络的决策分界线。输入命令：

```
net.adaptParam.passes = 3;
linehandle=plotpc(net.IW{1},net.b{1});
for a = 1:25
   [net,Y,E] = adapt(net,P,T);
   linehandle = plotpc(net.IW{1},net.b{1},linehandle);  drawnow;
end;
```

绘出图形如图 8-16 所示。

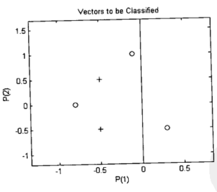


图 8-16　输入向量与期望响应样本图

查看误差向量 $\boldsymbol{E}$ 的值，为[1,1,0,0,-1]，没有达到 0。这表明，经过 25 次 adapt 训练后，网络仍然不能将输入样本正确地区分开来，这一点从图中的决策分界线也可以看出来。

实际上，对于这种无法线性划分的输入样本，感知器网络是无能为力的，这也是单层感知器神经网络的局限性所在。

8.6 小结

本章将首先介绍感知器网络结构以及其学习算法，然后介绍如何利用 MATLAB 神经网络工具箱函数生成、仿真和训练一个感知器神经网络，讨论单层感知器的局限性，最后给出 MATLAB 仿真实例作为学习的辅助参考。



第 9 章　线性神经网络

本章所要介绍的线性神经网络与感知器神经网络在网络结构上比较相似，也是最简单的神经网络的一种，最早期的代表是 20 世纪 50 年代由 Widrow 和 Hoff 提出的自适应线性元件（Adaptive Linear Element，Adaline）。只是线性神经网络采用的传递函数是线性函数而不是硬限值函数。这使得网络的输出可以取任何值，而不是像感知器一样只能够取 0 或者 1。而由于采用了线性函数作为传递函数，线性神经网络和感知器网络一样，也只能够适用于线性分类问题。

线性神经网络的设计目标，也是通过对输入向量赋予不同的权值和偏差值，对于给定的输入产生出符合期望的网络响应。在输出向量和目标向量之间的差值即为误差。通常，我们需要寻找的是这样一组权值和偏差，它能够使得网络输出相对于期望响应的均方误差达到最小。

对于线性系统，这个最小值是唯一存在的，因此这个问题是可解的。在大多数情况下，我们可以直接求解出一个线性网络，使得对于特定的输入向量和期望响应，其输出误差达到最小。而在另一些场合，直接求解可能会有些难度。但是幸运的是，我们总是可以通过由 Widrow 和 Hoff 提出的 LMS 算法（Least Mean Squares）对网络进行训练得到期望的最小均方误差网络。

9.1　线性神经网络结构

线性神经网络虽然在网络拓扑结构上与感知器网络相同，但是它以连续线性模拟量为输入，主要应用于自适应系统和连续可调过程。本节我们将对线性神经元和网络结构进行介绍。

9.1.1　线性神经元模型

图 9-1 给出了一个线性神经元的模型，该神经元具有 R 个输入变量 $\boldsymbol{p}_i$，对输入变量赋以不同的权重 $w_{1,i}$ 之后相加，并加入偏差 $\boldsymbol{b}$，其计算公式表示为：

$$\boldsymbol{n}=\sum_{i=1}^{R}\boldsymbol{p}_i w_{1,i}+\boldsymbol{b} \tag{9.1}$$

线性神经元的结构与我们在上一章介绍的感知器神经元的结构相同，唯一不同的是线性神经元是采用线性（purelin）函数作为网络传递函数，如图 9-2 所示。

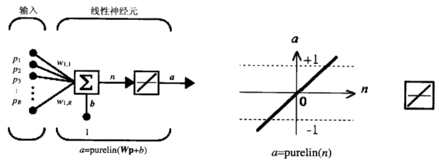


图 9-1 线性神经元模型示意图　　图 9-2 线性传递函数示意图

线性传递函数直接将输入的值作为神经元的输出结果，其计算公式如下：

$$\boldsymbol{a} = \text{purelin}(n) = \text{purelin}(\mathbf{Wp} + \boldsymbol{b}) = \mathbf{Wp} + \boldsymbol{b} \tag{9.2}$$

由于传递函数是线性的，因此线性神经元经过训练可以拟合一个线性函数，或者求得一个非线性函数的线性逼近，但是不能直接应用于非线性的计算。

9.1.2 线性神经网络结构

一个普通的线性神经网络的结构如图 9-3 所示，其中左图是分解形式表示的网络结构。可以看到，该线性神经网络为单层神经元结构，共包含 S 个神经元，有 R 个输入，$w_{i,j}$ 代表第 j 个输入对第 i 个神经元的权值。

图 9-3 的右图是以矩阵和向量形式表示的网络结构，输入向量 $\boldsymbol{p}$ 为 R×1 维，权值矩阵 $\boldsymbol{W}$ 为 S×R 维，偏差矢量 $\boldsymbol{b}$ 为 S×1 维，最终输出向量 $\boldsymbol{a}$ 为 S×1 维。

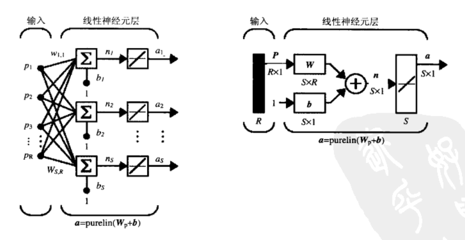


图 9-3 线性神经网络结构示意图

尽管上图表示的线性神经网络只具有单层网络结构，但是实际上，它与复杂的多层神

经网络结构是相同的。可以证明，对于每一个多层的线性神经网络，都能够找到一个单层的神经网络与之等效。

9.2 线性滤波器

在上一节中的线性神经网络上加上延迟线，就可以构成自适应线性滤波器，这是线性神经网络的一个非常重要的应用。作为线性神经网络的提出者，Widrow 对这一网络的应用也做了大量的研究工作，先后将其应用于天气预报、语音识别和图像分析等多种领域，这一节我们对线性滤波器进行介绍。

我们将为线性神经网络引入一个新的单元——延迟线（Tapped Delay Line），以便形成一个线性滤波器。假如加入了延迟线的线性神经网络如图 9-4 所示。这里信号从左边输入，经过 $N-1$ 个单元的延迟线传输，输出为 $N-1$ 维向量，形成当前时刻的神经元层输入信号，经过线性层后获得当前的滤波器输出。

$$\boldsymbol{a}(k)=\text{purelin}(\boldsymbol{W}\mathbf{p}+\boldsymbol{b})=\sum_{i=1}^{R}w_{1,i}\boldsymbol{a}(k-i+1)+\boldsymbol{b} \tag{9.3}$$

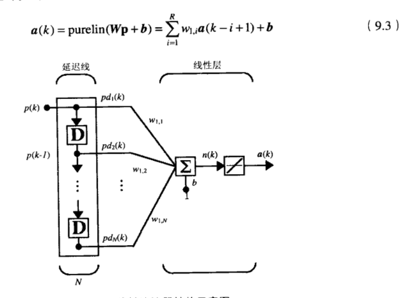


图 9-4　线性滤波器结构示意图

上面的网络构成了数字信号处理领域中所谓的有限冲击响应滤波器（FIR filter），通过遵循学习规则的训练过程，就可以获得一个线性自适应的滤波器，加入训练模块的自适应线性滤波器结构如图 9-5 所示。在后面我们将会进一步介绍。

9.3 线性神经网络学习规则

线性神经网络采用 Widrow-Hoff 提出的学习规则（W-H 学习规则）来修正权值矢量，采用 W-H 学习规则可以使得网络能够线性逼近一个函数式，从而进行模式联想。

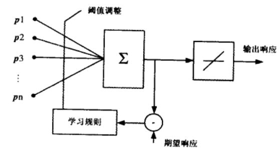


图 9-5 自适应线性滤波器结构示意图

9.3.1 均方误差

与感知器学习规则类似，LMS 算法同样是有监督学习的一种。学习规则是由一组输入向量和期望输出给出的，如下所示：

$$\{\boldsymbol{p}_1,\boldsymbol{t}_1\},\{\boldsymbol{p}_2,\boldsymbol{t}_2\},\ldots,\{\boldsymbol{p}_Q,\boldsymbol{t}_Q\}$$

其中，$\boldsymbol{p}$ 是训练时提供的输入向量；$\boldsymbol{t}$ 是期望的输出向量。将实际的网络输出与期望输出相比较，得到训练的均方误差，用公式表达如下：

$$mse=\frac{1}{Q}\sum_{k=1}^{Q}\boldsymbol{e}(k)^2=\frac{1}{Q}\sum_{k=1}^{Q}(\boldsymbol{t}(k)-\boldsymbol{a}(k))^2 \tag{9.4}$$

其中，$\boldsymbol{t}(k)$ 是 k 时刻的网络期望输出，$\boldsymbol{a}(k)$ 是此时刻对应的实际输出。LMS 网络训练的目标是使得上述均方误差达到最小。

幸运的是，线性神经网络的均方误差性能指标是一个二次函数，因此，此均方误差性能指标或者有一个全局最小解，或者有一个局部最小解，或者没有最小解，这取决于输入向量的特性。也就是说，输入向量的性质决定了对线性网络进行训练的解是否存在。

9.3.2 LMS 算法

LMS 算法又称为 Widrow-Hoff 学习规则，是基于近似最陡下降方法的学习规则。Widrow 和 Hoff 发现，可以通过计算每一步迭代过程中的方差来近似得到均方误差。如果将第 k 次迭代的方差对各个神经元的权值和偏差求偏导数，就可以得到：

$$\frac{\partial e^2(k)}{\partial w_{1,j}}=2e(k)\frac{\partial e(k)}{\partial w_{1,j}},\quad j=1,2,\ldots,R \tag{9.5}$$

$$\frac{\partial e^2(k)}{\partial b}=2e(k)\frac{\partial e(k)}{\partial b} \tag{9.6}$$

下面我们将偏差的表达式代入，就可以得到：

$$\frac{\partial e(k)}{\partial w_{1,j}}=\frac{\partial e[t(k)-a(k)]}{\partial w_{1,j}}=\frac{\partial[t(k)-(\mathbf{W}\mathbf{p}(k)+b)]}{\partial w_{1,j}} \tag{9.7}$$

$$\frac{\partial e(k)}{\partial w_{1,j}}=\frac{\partial[t(k)-(\sum_{i=1}^{R}w_{1,i}p_i(k)+b)]}{\partial w_{1,j}} \tag{9.8}$$

其中，$p_i(k)$ 是第 k 次迭代时的输入向量的第 i 个元素。经过化简，可以得到：

$$\frac{\partial e(k)}{\partial w_{1,j}}=p_j(k)\text{，}\quad \frac{\partial e(k)}{\partial b}=-1 \tag{9.9}$$

最后，得到权值和偏差矢量的改变量，分别为：

$$2\alpha e(k)p(k)\text{，}\quad 2\alpha e(k) \tag{9.10}$$

从而可以得到权值和偏差修改的方程，即 Wildrow-Hoff 学习算法，如下所示：

$$\boldsymbol{W}(k+1)=\boldsymbol{W}(k)+2\alpha e(k)\boldsymbol{p}^{\mathrm{T}}(k) \tag{9.11}$$

$$\boldsymbol{b}(k+1)=\boldsymbol{b}(k)+2\alpha \boldsymbol{e}(k) \tag{9.12}$$

其中，偏差 $\boldsymbol{b}$ 和误差 $\boldsymbol{e}$ 都以向量的形式给出。α 称为学习速率，如果 α 比较大的话，学习过程就会进行得相当快，但是如果学习过程过快，就可能造成求解过程的不稳定，反而会导致误差 $\boldsymbol{e}$ 的增加。

为了保证学习过程的稳定性，学习速率通常要比输入向量的自相关矩阵 $\boldsymbol{p}^{\mathrm{T}}\boldsymbol{p}$ 的最大本征值的倒数要小。

MATLAB 工具箱也提供了相应的函数 learnwh，能够进行相关的计算。此函数的计算公式表示为：

$$\mathrm{d}w=lr\cdot e\cdot p \tag{9.13}$$

$$\mathrm{d}b=lr\cdot e \tag{9.14}$$

其中（9.11）、（9.12）两式中的常数 2 被规划到了学习速率 lr 中去了。

MATLAB 工具箱还提供了另一个函数 maxlinlr，通过计算矩阵 $0.999\boldsymbol{p}^{\mathrm{T}}\boldsymbol{p}$ 的本征值得到最大的稳定学习速率。

9.4 线性神经网络的 MATLAB 实现

本节介绍 MATLAB 工具箱中关于线性神经网络的函数和这些函数的应用。

9.4.1 线性神经元生成

考虑一个具有两个输入的线性神经元，其结构如图 9-6 所示。

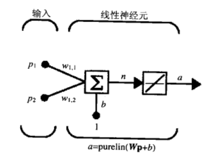


图 9-6　二输入线性神经元结构示意图

由于整个网络只有一个神经元，因此权值矩阵为行向量。网络输出表示为：

$$a = \text{purelin}(n) = \text{purelin}(\mathbf{Wp} + b) = \mathbf{Wp} + b \tag{9.15}$$

或者

$$a = w_{1,1}p_1 + w_{1,2}p_2 + b \tag{9.16}$$

如同感知器一样，线性神经网络的决策边界是由使得输出为 0 的输入向量决定的。即 a 为 0 时，由 $\mathbf{Wp} + b = 0$ 表示的直线，如图 9-7 所示。

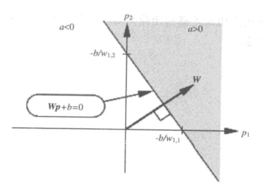


图 9-7　二输入线性神经元决策边界线

图 9-7 中，处于右上部分灰色区域中的输入向量通过线性网络后产生的输出结果是大于 0 的，而处于左下部分白色区域内的输入向量通过线性网络后得到的输出结果则小于 0。

因此，线性神经网络也可以用来对输入信号进行二类划分，但是同样只有在输入变量是线性可分的情况下才适用。在这一点上，线性神经网络与感知器神经网络具有同样的局限性。

调用函数 newlin 可以生成一个线性神经元，newlin 函数的调用格式如下：

net = newlin(***P***, *S*, ID, LR)

net = newlin(***P***, ***T***, ID, LR)

在第一种格式中：

- P 为 $R \times Q$ 阶矩阵，代表输入向量的取值范围，Q 代表输入向量个数；
- S 为输出向量的元素个数；
- ID 为输入延迟向量，默认值为[0]，没有延迟；
- LR 代表学习速率，默认值为 0.01。

而在第二种格式中：

- T 为 $S \times Q$ 阶矩阵；
- S 为输出向量的元素个数。

函数执行的返回结果是一个线性神经元对象。

【例 9-1】 线性神经元生成实例。本例生成一个二输入的线性神经元并仿真。

解：这里我们调用 newlin 函数以生成线性神经元，给定典型的输入向量为[-1,-1]和[1,1]，以及典型的输出向量为[-1, 1]。对于一个实际问题，通常输入和输出使用的都是实数向量。输入命令：

```
net = newlin([-1 1; -1 1],[-1 1]);
```

以上命令执行的结果是生成了一个线性神经网络对象。注意这里参数中给定的所谓典型的输入和输出向量并不是直接对应的，而是一个量级上的概念，便于生成适当的网络权值和偏差。

默认情况下，网络权值和偏差都初始化为 0。我们可以通过下面的语句查看当前的网络权值和偏差。输入命令：

```
W = net.IW{1,1}
```

得到：

```
W =     0     0
```

输入命令：

```
b= net.b{1}
```

得到：

```
b =     0
```

我们也可以对权值和偏差赋予我们期望的值，例如将权值分别赋值为 2 和 3。输入命令：

```
net.IW{1,1} = [2 3];
W = net.IW{1,1}
```

得到：

```
W =     2     3
```

再检查偏差 b 的值。输入命令：

```
net.b{1} = [-4];
```

就得到：

```
b = net.b{1}
b =     -4
```

接下来我们通过给定的输入向量对生成的网络进行仿真。首先定义输入向量 ***p***。输入命令：

```
p = [5;6];
```

再利用 sim 函数进行网络仿真。输入命令：

```
a = sim(net,p)
```

得到网络仿真结果为：

```
a =    24
```

在本例中，我们通过工具箱提供的 newlin 函数建立了一个线性神经网络对象，并调整了内部参数的赋值，应用 sim 函数进行了网络的仿真。

9.4.2 线性神经网络生成

线性神经网络与其他种类较复杂的网络不同的是，在输入和输出目标向量已知的情况下，线性神经网络可以通过函数直接产生。利用函数 newlind，我们可以设计出特定的网络权值和偏差，以达到均方误差最小化的目标。

newlind 函数的调用格式如下：

net = newlind(***P***, ***T***, Pi)

其中：

- ***P*** 为 $R\times Q$ 阶矩阵，Q 为输入向量个数；
- ***T*** 为 $S\times Q$ 阶矩阵，S 为输出向量的元素个数，输出向量个数也为 Q；
- Pi 是 1×ID 维的初始输入延迟状态。

函数执行的结果是返回一个单层线性神经网络。对于指定的输入、输出向量 ***P*** 与 ***T***，此线性神经网络具有最小的均方误差。

【例 9-2】 线性神经网络生成实例。假定输入向量和目标向量分别为 ***P*** = [1 2 3]，***T***= [2.0 9.1 5.9]，利用 newlind 函数设计相应的线性神经网络对象。

解： 先定义输入输出向量，然后调用 newlind 函数。在 MATLAB 命令行输入：

```
P = [1 2 3];
T= [2.0 4.1 5.9];
net = newlind(P,T);
```

这样我们得到了一个线性神经网络对象。通过 sim 函数对其进行仿真，输入命令：

```
Y = sim(net,P)
```

得到仿真结果：

```
Y =    2.0500    4.0000    5.9500
```

我们发现，网络仿真输出结果与期望的目标响应是非常接近的。利用 newlind 函数也可以生成带有延迟线的线性网络结构，这将在下一节中介绍。

9.4.3 线性滤波器生成

对于有延迟线的线性神经网络，即线性滤波器，同样可以通过 newlind 函数来生成，下面举例进行介绍。

【例 9-3】 线性滤波器生成实例 1。假定网络输入向量和目标向量分别为 ***P*** = [1,2,1,3,3,2]，***T*** = [5,6,4,20,7,8]，延迟状态为 Pi = [1,3]，利用 newlind 函数生成相应的线性滤波器网络。

解：定义输入输出向量。在命令行中输入：

```
P = {1 2 1 3 3 2};
T = {5 6 4 20 7 8};
```

通过下面的语句指定网络输入的初始延迟：

```
Pi = {1 3};
```

此延迟设置表示神经元有两点输入，分别由输入向量经过 1 个单位和 3 个单位的延迟而得到。我们可以利用 newlind 函数来生成相应的线性滤波器网络。在 MATLAB 命令行中输入命令：

```
net = newlind(P,T,Pi);
```

即生成了需要的线性滤波器网络。利用 sim 函数进行网络输出的仿真：

```
Y = sim(net,P,Pi)
```

得到结果为：

```
Y = [2.7297] [10.5405] [5.0090] [14.9550] [10.7838] [5.9820]
```

可以看到，仿真得到的网络响应并不与期望响应完全一致，但是相对来说还是比较接近的。产生这么大的误差的原因，是因为所采用的延迟线太短，如果增加延迟线的长度，生成的线性滤波器对于目标响应的逼近性能上将会有大幅的提高。

不管怎样，利用 newlind 函数生成的线性滤波器网络保证了输出相对于期望响应的均方误差最小化。

【例 9-4】 线性滤波器生成实例 2。生成一个具有两个输入、两个输出的线性滤波器网络，使得输入向量分别为 P1 = [1,2,1,3,3,2]，P2 = [1,2,1,1,2,1]的情况下，目标向量分别为 T1 = [5.0,6.1,4.0,6.0,6.9,8.0]，T2 = [11.0,12.1,10.1,10.9,13.0,13.0]；并且初始延迟状态分别为 Pi1 = [1,3,0]，Pi2 = [2,1,2]。

解：① 首先定义上述向量。输入命令：

```
P1 = {1 2 1 3 3 2}; Pi1 = {1 3 0};
```

```
P2 = {1 2 1 1 2 1}; Pi2 = {2 1 2};
T1 = {5.0 6.1 4.0 6.0 6.9 8.0};
T2 = {11.0 12.1 10.1 10.9 13.0 13.0};
```

然后应用 newlind 函数生成线性滤波器网络。输入命令：

```
net = newlind([P1; P2],[T1; T2],[Pi1; Pi2]);
```

② 利用 sim 函数仿真。输入命令：

```
Y = sim(net,[P1; P2],[Pi1; Pi2]);
```

利用下列语句查看两个输出的最终结果：

```
Y1 = Y(1,:)
Y2 = Y(2,:)
```

输出结果为：

```
Y1 =  [5.0000]    [6.1000]    [4]   [6]    [6.9000]    [8.0000]
Y2 =  [11.0000]   [12.1000]   [10.1000]    [10.9000]   [13.0000]   [13]
```

在此例中我们看到，生成的线性滤波器网络能够很好的符合期望响应。与【例 9-3】相比较，这里滤波器延迟线的长度增加 1，滤波器的性能得到了很大的提升。这也体现了数字信号处理中 FIR 滤波器阶数对性能的影响。

9.4.4 线性神经网络训练

尽管线性神经网络的设计可以直接进行，这一节里我们仍将通过训练的方式，使得一个神经元对于给定的输入获得指定的输出。

这一节我们需要用到的新的 MATLAB 工具箱函数包括 errsurf、plotes、maxlinlr、plotep、plotperf。首先我们对这些相关函数的调用格式进行介绍，再通过实例进行说明。

1．errsurf

函数 errsurf 用于计算一个单输入神经元的误差表面矩阵。其调用格式如下：

errsurf(P, T, WV, BV, F)

其中：

- ***P*** 为 1×*Q* 阶的输入向量；
- ***T*** 为 1×*Q* 阶的目标输出向量；
- WV 为可取权值 *W* 组成的行向量；
- BV 为可取偏差 *B* 组成的行向量；
- *F* 为传递函数。

函数最后返回一个由 WV 和 BV 决定的误差矩阵。

2．plotes

函数 plotes 绘出一个单输入神经元的误差表面。其调用格式如下：

plotes(WV, BV, ES, *V*)

其中：

- WV 为可取的权值 **W** 组成的行向量；
- BV 为可取的偏差 **B** 组成的行向量；
- ES 是由 WV 和 BV 决定的误差矩阵，可以通过上面的 errsurf 函数计算得到；
- **V** 是三维图采用的观察角度，默认的观察角度为[-37.5,30]，其中前一个为方位角，后一个数值为俯仰角。

函数的执行结果是以三维曲面和等高线的形式，绘制出神经元的误差表面图。

3. maxlinlr

函数 maxlinlr 用来计算线性神经元层的最大学习速率，其调用格式如下：

lr = maxlinlr(P)

lr = maxlinlr(P, 'bias')

其中 **P** 为 $R \times Q$ 阶的输入向量矩阵。

4. plotep

函数 plotep 用来在 plotes 得到的误差曲面和等高线图上绘出当前网络的误差性能。其调用格式如下：

h = plotep(***W***, ***B***, ***E***)

其中：

- **W** 为当前的网络权值；
- **B** 为当前的偏差向量；
- **E** 为误差矩阵。

5. plotperf

函数 plotperf 用来显示网络学习过程中网络性能随迭代次数的变化。其调用格式为：

plotperf(TR, goal, name, epoch)

其中：

- TR 为 train 训练函数返回的训练记录对象；
- goal 是训练目标值，默认值为 NaN；
- name 是网络函数名称，默认值为' '；
- epoch 为目前的训练的迭代次数。

下面用实例进行介绍。

【例 9-5】 线性神经网络训练实例。给定一维输入向量 ***p***= [−6.0, −6.1, −4.1, −4.0, +4.0, +4.1, +6.0, +6.1]，期望目标输出 ***t*** = [+0.0, +0.0, +.97, +.99, +.01, +.03, +1.0, +1.0]，试观察线性神经网络对于此输入和目标输出的误差表面，并设计一个网络对象进行学习，观察其训练情况。

解： 此问题用单个神经元就可以完成求解。

① 首先定义输入和期望输出。输入：

```
p = [-6.0 -6.1 -4.1 -4.0 +4.0 +4.1 +6.0 +6.1];
t = [+0.0 +0.0 +.97 +.99 +.01 +.03 +1.0 +1.0];
```

② 给定权值和偏差的取值范围。输入：

```
wv = -1:.1:1; bv = -2.5:.25:2.5;
```

然后通过 errsurf 和 plotes 画出误差曲面，输入命令：

```
es = errsurf(p,t,wv,bv,'logsig');
plotes(wv,bv,es,[60 30]);
```

执行结果如图 9-8 所示。

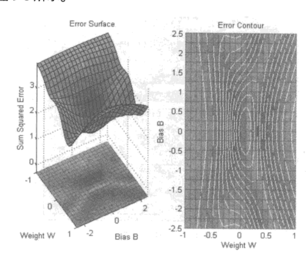


图 9-8 线性网络误差曲面与等高线图

其中，左图是以三维图的格式绘出的误差曲面，右图是以等高线的格式绘出的误差曲面。可以从中看到在权值和偏差可取值范围之内，误差曲面确实包含一个全局最小点。

③ 首先计算出在此输入向量下能够得到的线性网络的最大学习速率，然后定义一个网络对象，采用 90%的最大速率进行学习，并进行观察。输入命令：

```
maxlr = 0.90*maxlinlr(p,'bias');
```

输出结果为：

```
maxlr =    0.0042
```

这个学习速率实际上是比较低的。创建一个线性网络对象，输入命令：

```
net = newlin([-2 2],1,[0],maxlr);
```

其中 newlin 的调用格式我们已经在上面介绍过。在此例中，参数[-2, 2]为输入的取值范围，输出个数为 1，网络没有延迟，采用的最大学习速率为 maxlr。

④ 下面修改网络对象训练的属性，将训练目标值设定为 0.001。输入：

```
net.trainParam.goal = .001;
```

应用 train 函数对网络进行训练。输入命令：

```
[net,tr] = train(net,P,T);
```

其中返回值 net 是每一步训练的结果，tr 是一个结构体，保存了训练过程的信息。命令执行的时候，弹出了训练工具窗口，函数执行结果如图 9-9 所示。

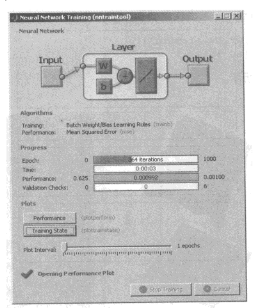


图 9-9　训练工具窗口

随着训练过程的进行，epoch 进度条逐渐增加，显示迭代次数。默认的最大迭代次数为 1000 次。此例中可以看到，迭代进行了 364 次，网络输出才达到设定的训练目标值，即均方误差降到我们设定的 0.001 以下。

对此我们可以理解，对于比较长的输入、输出向量，达到比较小的误差是需要一定量的迭代次数的。

⑤ 单击窗口上的"Performance"按钮，我们可以查看均方误差随迭代次数的变化情况，如图 9-10 所示。

图中实线是训练的均方误差值，点线是训练目标值。注意到，左边的坐标值采用的是指数刻度。从图中可以看到，网络输出的均方误差值随着训练的进行呈指数下降，最终下降到 0.001 所用的迭代次数为 364。在后面的例子中我们将会看到，这条误差性能线也可以通过函数 plotperf 得到。

这个例子中，在给定输入向量和期望输出向量的情况下，我们首先查看了权值和偏差取值范围内的线性网络的误差表面情况，然后在给定的最大学习速率下，利用 train 函数完成了一个线性网络的训练，并且观察了训练过程中均方误差随迭代步数的变化，从而看到了 Widrow-Hoff 学习规则实际应用的情况。

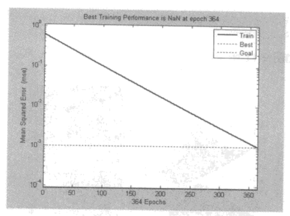


图 9-10 训练中均方误差 mse 随迭代次数而变化

9.5 线性网络的局限性

尽管相对于感知器网络而言，线性网络的取值区间不再只是二值范围，但是线性网络仍然只能够表征输入与输出向量之间的线性关系，因此对于一些问题线性神经网络仍然是无法求解的。

不过，即使在无法获得精确解的情况下，只要学习速率取得足够小，线性网络仍然能够将输出的均方误差最小化。这种网络结构能够找到输入与输出之间最佳的近似线性逼近，具有这种性质是因为线性网络的误差曲面是一个抛物面，因为抛物面总是只有一个最小值的，因此利用最大梯度算法（例如 LMS 算法）总能够得到最小值解。

线性网络还有其他的一些限制，例如对超定系统、不定系统、线性相关向量的情况，这一节将对这些内容加以介绍。

9.5.1 非线性系统

对于非线性系统，线性神经网络无法获得精确解，只能够得到其线性逼近。这意味着模型的偏差会带来比较大的误差。下面以实例进行说明。

【例 9-6】 非线性系统神经元拟合实例。给定输入向量和期望输出向量分别为 $\boldsymbol{P}$ = [+1.0, +1.5, +3.0, −1.2]，$\boldsymbol{T}$ = [+0.5, +1.1, +3.0, −1.0]，这两个向量是经过设计，无法用 $\boldsymbol{WP}+\boldsymbol{B}=\boldsymbol{T}$ 的线性关系进行表出的。接下来利用 LMS 算法训练一个线性神经元来对此输入和输出进行拟合，看看会得到什么结果。

解： ① 定义输入向量和目标输出响应。输入命令：

```
P = [+1.0 +1.5 +3.0 -1.2];
T = [+0.5 +1.1 +3.0 -1.0];
```

② 通过 errsurf 和 plotes 画出误差曲面。输入命令：

```
w_range =-2:0.4:2;  b_range = -2:0.4:2;
```

```
ES = errsurf(P,T,w_range,b_range,'purelin');
plotes(w_range,b_range,ES);
```

命令执行结果如图 9-11 所示。

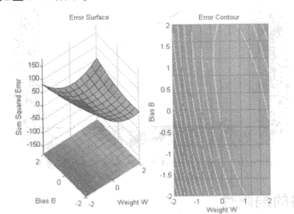


图 9-11　非线性输入输出的误差曲面与等高线图

③ 计算线性网络的最大学习速率，并生成线性网络。输入命令：

```
maxlr = maxlinlr(P,'bias');
net = newlin([-2 2],1,[0],maxlr);
```

④ 为了显示出训练过程中每一步生成的网络的误差性能在误差曲面上的变化情况，我们每次运行一个周期就调用一次 plotep 函数在误差表面上绘一次图。

由于 newlin 生成的网络默认的最大训练迭代次数设置为 1000，因此下面需要设置网络最大训练迭代次数为 1，这样保证训练过程中每一步可以中止。对此时网络对象进行绘图显示。输入命令：

```
net:train-param.epochs=1;
```

⑤ 完成整个训练的过程，并绘图。各个点之间用蓝线连接起来，形成网络均方误差在误差性能面上运动的轨迹。输入命令：

```
h=plotep(net.IW{1},net.b{1},mse(T-sim(net,P)));   %绘出误差性能起点
[net,tr] = train(net,P,T);                        %进行单步训练
r = tr;                                           %保存训练过程对象
epoch = 1;                                        %设定变量保存训练迭代步数
while epoch < 15                                  %开始循环，最大迭代次数为 15
   epoch = epoch+1;                               %迭代步数加 1
   [net,tr] = train(net,P,T);                     %训练
   if length(tr.epoch) > 1
      h = plotep(net.IW{1,1},net.b{1},tr.perf(2),h);%画出每一步误差性能点
      r.epoch=[r.epoch epoch];                    %将训练过程中的迭代次数和误差保存到对象 r 中
      r.perf=[r.perf tr.perf(2)];
      r.vperf=[r.vperf NaN];
```

```
      r.tperf=[r.tperf NaN];
    else
      break
    end
  end
  tr=r;                                   %最后一步重新赋值，将中间变量 r 的值赋给 tr
```

运行上面代码，作出图像，得到结果如图 9-12 所示。

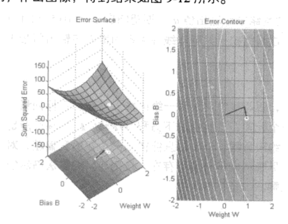


图 9-12 训练过程中网络误差性能的变化

可以看到，实线显示了整个训练过程中误差性能在误差表面上的移动情况，最终完成 15 次迭代之后，误差性能点移动到了白色点位置。

⑥ 下面我们利用函数 plotep 绘出误差性能随着迭代次数的变化情况。如果是一次进行多步迭代训练的话，这个功能也可以如上例通过训练工具窗口上的“Performance”按钮实现，但是这里因为采用了单步执行，因此需要利用下面命令的方式实现。输入命令：

```
plotperf(tr,net.trainParam.goal);
```

此命令绘出能显示误差随迭代次数的变化曲线，以及训练期望得到的误差性能目标线，如图 9-13 所示。

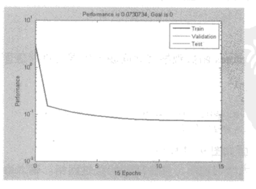


图 9-13 误差性能随训练迭代次数而变化

从图中我们可以看到，训练结束时网络的误差性能值为 0.073。对预定输入进行仿真，输入命令：

```
a = sim(net, P)
```

得到网络输出为：

```
a =    0.8500    1.3114    2.6958   -1.1803
```

可以看到，训练后的网络响应输出值 ***a*** 与期望响应 T = [+0.5, +1.1, +3.0, −1.0]还是有一定的差距的，这是用线性网络拟合具有非线性关系的输入与输出所带来的误差。

9.5.2 超定系统

考虑这样一种情况，假定线性网络有一维的输入向量和目标向量，但是每一个输入向量和目标向量都含有四个元素，这种情况下方程 $WP+B=T$ 的解可能是不存在的，因为在这个问题下，存在四个约束方程，但是却只能得到一个权值和偏差，对于这种问题，我们称它为超定的。然而，尽管精确解可能不存在，应用 LMS 算法仍然能够使得均方误差最小化，以保证获得最优解。

9.5.3 不定系统

对于不定情况，可以考虑一个单输入线性神经元，仅仅通过两个一维的输入和期望输出向量对其进行训练，且两个输出均只包含一个元素，例如 P = [1.0]，T = [0.5]。

这种情况下，由于只有一个约束方程，但是却有权值与偏差两个待定变量，待定变量的个数多于约束方程的个数，这会导致问题具有无限个解，也就是所谓的不定系统问题。下面举例说明。

【例 9-7】 不定系统神经元拟合实例。给定输入向量和期望输出向量分别为 P = [+1.0]，T = [+0.5]，我们知道此线性问题的对应解的个数为无限个，利用 LMS 算法训练一个线性神经网络来对此输入和输出进行拟合。

解： ① 首先定义输入和输出向量。输入命令；

```
P = [+1.0];
T = [+0.5];
```

② 利用 errsurf 与 plotes 函数观察其此问题下的误差性能表面。输入命令：

```
w_range = -1:0.2:1;
b_range = -1:0.2:1;
ES = errsurf(P,T,w_range,b_range,'purelin');
plotes(w_range,b_range,ES);
```

得到的误差性能表面如图 9-14 所示。

③ 计算网络的最大学习速率，并生成初始线性网络。输入命令：

```
maxlr = maxlinlr(P,'bias');
```

```
net = newlin([-2 2],1,[0],maxlr);
```

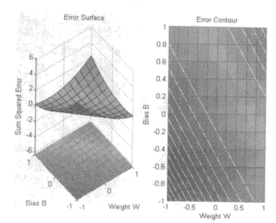


图 9-14 不定问题的误差曲面与等高线图

④ 设定网络的训练目标为 1e-10。要注意，这是一个非常小的数，相对来说，精度要求是非常高的。输入命令：

```
net.trainParam.goal = 1e-10;
```

⑤ 同上例，我们设定单次训练的迭代次数为 1，然后逐次画出误差性能表面上点的移动情况。这里我们的循环过程没有设定最大值，这样的结果是，迭代会一直进行下去，一直到网络训练达到目标为止。输入命令如下：

```
net.trainParam.epochs = 1;                          %设定单次训练的迭代次数为 1
h=plotep(net.IW{1},net.b{1},mse(T-sim(net,P)));     %绘出误差性能起点
[net,tr] = train(net,P,T);
r = tr;                          %保存训练过程对象
epoch = 1;
while true                       %迭代过程一直持续，直到满足训练目标为止
   epoch = epoch+1;
   [net,tr] = train(net,P,T);
   if length(tr.epoch) > 1
      h = plotep(net.IW{1,1},net.b{1},tr.perf(2),h); %画出每步的误差性能点
      break
   end
end
tr=r;                            %最后一步重新赋值，将中间变量 r 的值赋给 tr
```

运行上面代码之后，曲面上绘出了每一步的误差性能点，如图 9-15 所示。我们看到，仅仅进行了一次迭代就退出了循环。

⑥ 作出误差性能随迭代次数的变化图。输入命令：

```
plotperf(tr,net.trainParam.goal);
```

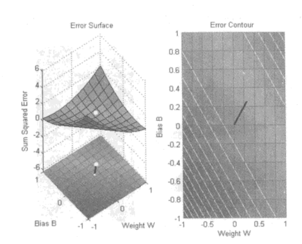


图 9-15　训练中网络的误差性能点在误差性能曲面的移动

绘出图形如图 9-16 所示。

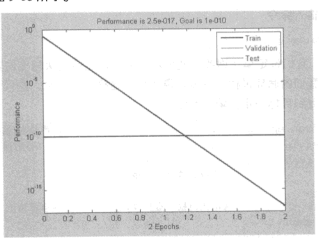


图 9-16　训练中网络的误差性能随迭代次数的变化

可以清楚地看到，迭代仅仅进行了一步，误差性能就立刻达到了 1e-10 以下。对于这么高的精度目标，一步迭代就能达到，这就是由于不定问题有无限多个精确解造成的，因此在误差性能表面上，很容易就能够找到适应于此问题的解。

同样我们可以对生成的网络进行仿真。输入下列语句：

```
a = sim(net,P);
```

得到：

```
a =   0.5000
```

可以看到，此训练生成的网络能够得到精确的期望响应。

⑦ 为了进行对比，我们可以用另一种方法来生成一个线性网络，比较一下这两种方法的解。

前面介绍过，对于给定的输入向量和期望输出向量，利用工具箱中的函数 newlind 可以直接生成适应于此问题的线性网络。下面我们利用这个函数生成一个网络，然后将其与上面训练产生的网络进行比较。

首先关闭图 9-16 绘图窗口，然后输入命令：

```
solvednet = newlind(P,T);   %用 newlind 直接求解产生一个新的网络
hold on;                    %保持打开的 4-15 绘图窗口
plot(solvednet.IW{1,1},solvednet.b{1},'ro') %在原图上用红点标出此网络性能点
hold off;
```

执行上述命令得到的图形如图 9-17 所示。

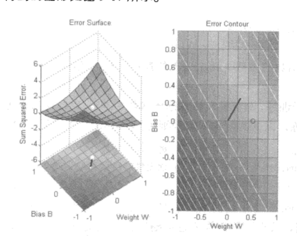


图 9-17 用 newlind 产生的网络与训练产生的网络比较

图中新生成的网络性能点用圆圈表出。可以看到，它与训练产生的网络权值和偏差都不相同，然而这两组网络都是此问题的精确解。也就是说，适合此问题的解是不唯一的，这就是方程组不定造成的结果。

9.5.4 线性相关向量

通常确定一个线性网络是否适合于求解待定问题的方法很简单。设神经网络输入为 S 个，每个输入为 R 元向量，则其对应线性方程的自由度数目为 $S*R+S$，也就是网络的待定权值和偏差的个数，而约束方程的个数为 Q，这是由输入向量和输出向量的元素个数决定的，如果 $S*R+S=Q$ 的话，就可以求得精确解。

然而，如果没有误差的输入向量之间存在线性相关性的话，约束方程的个数将小于自由度的个数，从而导致方程无法求得符合条件的精确解。

【例 9-8】 线性相关向量拟合实例。给定输入向量和期望输出向量分别为 $\boldsymbol{P}=[1.0, 2.0,$

3.0; 4.0, 5.0, 6.0]，*T* = [0.5, 1.0, −1.0]，训练一个二输入线性神经网络来对此其进行拟合。

解： ① 首先定义输入和输出向量。输入命令：

```
P = [ 1.0   2.0   3.0;
    4.0   5.0   6.0];
T = [0.5 1.0 -1.0];
```



对于 *P*，第二列输入向量恰好是第一列与第三列输入向量之和的 1/2，这表示输入向量是线性相关的；然而对于 *T*，列向量之间并没有同样的线性关系。这样的结果会导致线性神经元找不到合适的解。

② 计算对于此输入向量网络可具有的最大学习速率，并生成初始线性网络。输入如下命令：

```
maxlr = maxlinlr(P,'bias');
net = newlin([0 10;0 10],1,[0],maxlr);
```

其中，newlin 函数的参数，两个输入的最小和最大值都设定为[0 10]，输出个数设定为 1，输入延迟为 0。

③ 对网络进行训练，设定最大训练迭代次数为 500，目标均方误差为 0.001，利用 train 函数对网络进行训练，默认的网络学习规则为 Widrow-Hoff 规则。输入命令：

```
net.trainParam.epochs = 500;  % 最大迭代次数.
net.trainParam.goal = 0.001;  % 误差性能目标
[net,tr] = train(net,P,T);
```

从弹出 nntraintool 窗口中我们看到，训练完成了全部 500 次迭代。单击弹出窗口上的“Performance”按钮，会得到网络误差性能随迭代次数变化的曲线，如图 9-18 所示。

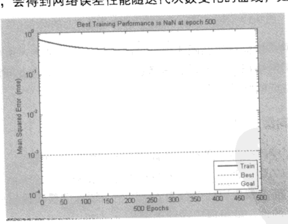


图 9-18　网络误差性能随迭代次数而变化

从图 9-18 中可以明显地看到，经过 500 次迭代，网络误差性能仍然在 0.1 以上，远没有达到期望的 0.001 的水平。这是因为输入向量线性相关，而期望响应并不具有相同的线

性关系。

④ 下面测试训练后的网络对第一个输入[1;4]的响应。输入命令：

```
p = [1.0; 4];
a = sim(net,p)
```

得到输出结果为：

```
a =    0.8971
```

与期望的 T(1) = 0.5 相差很远，也说明网络确实没有达到我们的期望。

9.5.5 学习速率过大

我们前面介绍过，对于线性神经网络，总可以通过 Widrow-Hoff 的 LMS 学习算法找出权值和偏差的最小均方误差解，但前提是学习速率足够小。

当学习速率过大时，大到超过 maxlinlr 求出的最大学习速率的时候，可能会导致迭代过程中均方误差反常增大，这会导致网络学习时间增加，或者会无法收敛到最小均方误差解。下面对此用实例说明。

【例 9-9】 学习速率的影响实例。按下面给定的输入向量和期望响应，用大于 maxlnlr 结果的学习速率对网络进行训练，检查网络训练的结果。

解： ① 定义输入和输出向量。输入命令：

```
P = [+1.0 -1.2];
T = [+0.5 +1.0];
```

② 观察此输入和输出向量的误差曲面。输入命令：

```
w_range = -2:0.4:2;
b_range = -2:0.4:2;
ES = errsurf(P,T,w_range,b_range,'purelin');
plotes(w_range,b_range,ES);
```

作出的误差曲面图如图 9-19 所示。

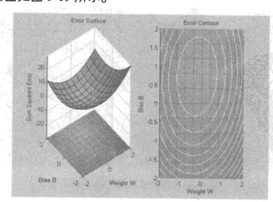


图 9-19 误差曲面与等高线图

③ 计算最大学习速率，并生成初始线性网络，这里网络采用的学习速率为 maxlr 的 2.25 倍。输入命令：

```
maxlr = maxlinlr(P,'bias');
net = newlin([-2 2],1,[0],maxlr*2.25);
```

④ 对网络进行训练，设定最大训练迭代次数为 20，利用 train 函数对网络进行训练，此处对训练中网络的误差点在曲面上的移动轨迹进行显示。与【例 9-6】相同，将训练设置为单步执行。输入命令：

```
net.trainParam.epochs = 1;
net.trainParam.show = NaN;
h=plotep(net.IW{1},net.b{1},mse(T-sim(net,P)));
[net,tr] = train(net,P,T);
r = tr;
epoch = 1;
while epoch < 20
   epoch = epoch+1;
   [net,tr] = train(net,P,T);
   if length(tr.epoch) > 1
      h = plotep(net.IW{1,1},net.b{1},tr.perf(2),h);
      r.epoch=[r.epoch epoch];
      r.perf=[r.perf tr.perf(2)];
      r.vperf=[r.vperf NaN];
      r.tperf=[r.tperf NaN];
   else
      break
   end
end
tr=r;
```

得到的误差点在曲面的移动轨迹如图 9-20 所示。

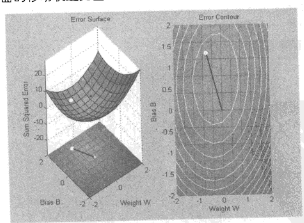


图 9-20　误差性能点的移动情况

可以看到，误差性能点并未移动到期望的误差曲面最低点。

⑤ 下面作出误差性能随迭代次数的变化图。输入命令：

```
plotperf(tr,net.trainParam.goal);
```

绘出图形如图 9-21 所示。

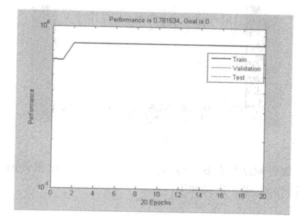


图 9-21 误差性能随迭代次数而变化

可以看到，随着训练的进行，网络的误差性能非但没有减小，反而增大了，这就是训练速率过大造成的后果。

9.6 线性神经网络设计实例

线性神经网络可以应用于系统辨识、信号预测、自适应滤波和控制等诸多方面。这一节我们将对其应用实例进行介绍。

9.6.1 线性预测

【例 9-10】 线性神经网络预测实例。假设输入信号为采样间隔为 0.01s，持续时间 2s 的正弦信号，试构建合适的线性神经网络，通过某一时刻的前 6 个输入信号采样值预测下一个信号的值。

解： ① 定义采样时间序列和目标信号，以及信号长度。输入命令：

```
Time = 0:0.01:2;
T = sin(Time*8*pi);
Length = length(T);
```

输入下面的命令，作出信号曲线：

```
plot(Time, T);
xlabel('时间');
ylabel('目标信号');
```

得到目标信号波形如图 9-22 所示。

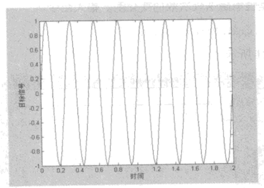


图 9-22　目标信号

② 我们采用的线性网络具有 6 个输入，1 个输出，输入向量分别是 ***T*** 延迟 1、2、3、4、5、6 个采样点得到的结果。定义如下：

```
P=zeros(6,Length);
for i=1:6
   P(i,i+1:Length)=T(1:Length-i);
end
```

得到输入向量矩阵 ***P*** 的定义。

③ 下面直接利用 newlind 函数来设计 ***P***，***T*** 所对应的网络。输入命令：

```
net = newlind(P,T);
```

④ 接下来利用 sim 函数仿真，测试其网络性能。输入命令：

```
a = sim(net,P);
plot(Time,a,'r');
xlabel('时间');
ylabel('预测输出');
```

得到网络的预测输出图如图 9-23 所示。

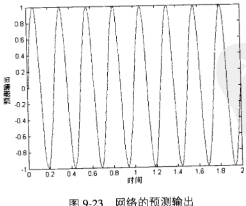


图 9-23　网络的预测输出

⑤ 下面作出误差图。输入如下命令：

```
e=T-a;
figure,
plot(Time,e);
xlabel('时间');
ylabel('预测误差');
```

得到的误差曲线如图 9-24 所示。

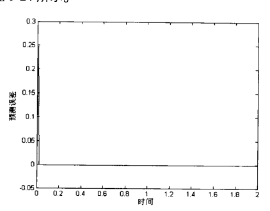


图 9-24 网络的预测误差曲线

从图中可以看到，仅仅在刚开始的很短的时间内，网络的预测误差比较大，经过训练预测误差很快降到接近 0 值，网络实现了对目标进行预测的功能。

在非线性问题的预测中，线性网络也能够给出最小的线性预测误差，给出问题的最佳线性逼近。

9.6.2 自适应滤波噪声抵消

【例 9-11】 线性神经网络自适应滤波实例。本例我们利用线性网络 adapt 函数来对输入信号进行自适应滤波，获得期望的输出响应。

解：① 设定输入信号 ***P*** 与期望输出 ***T***，采样率为 100Hz，持续采样 2.5s。输入如下命令：

```
time = 1:0.01:2.5;
X = sin(sin(time).*time*10);
P = con2seq(X);
T = con2seq(2*[0 X(1:(end-1))] + X);
```

其中 con2seq 函数的作用是将输入序列转化为 adapt 函数所需要的序列信号格式。

② 作出输入向量和期望响应图。输入如下命令：

```
plot(time,cat(2,P{:}),time,cat(2,T{:}),'--')
title('输入信号与期望响应')
xlabel('时间')
legend({'输入','期望响应'})
```

得到的图形如图 9-25 所示。

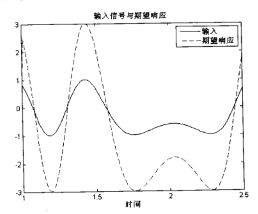


图 9-25　输入信号与期望响应图

③ 利用 newlin 函数生成线性网络。设定网络输入最小和最大值分别为–3 和 3，包含一个神经元，网络输入首先通过延迟线，延迟参数设定为[0 1]，表示由两个值进行加权，这两个进行加权运算的值分别是当前输入信号和延迟 1 个采样间隔的结果。输入命令：

```
net = newlin([-3 3],1,[0 1],0.1);
```

④ 利用 adapt 函数对输入信号进行自适应滤波，adapt 函数需要的参数是转化为序列信号格式的 ***P***，***T*** 向量，然后将网络自适应滤波的输出、期望响应、以及误差画在图上。输入命令如下：

```
[net,Y,E,Pf]=adapt(net,P,T);
figure,
plot(time,cat(2,Y{:}),'b', ...
   time,cat(2,T{:}),'.', ...
   time,cat(2,E{:}),'+',[1 2.5],[0 0],'k')
legend({'输出','期望响应','误差'})
```

结果如图 9-26 所示。

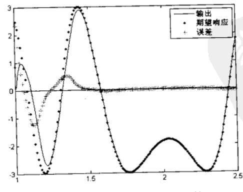


图 9-26　输入、期望响应以及误差图

在图 9-26 中网络的实际输出用曲线表示，而期望响应用圆点表示，误差曲线用加号“+”表示。可以看到，线性网络可以很好地实现对网络输出的线性逼近，最终学习的结果是使得均方误差值达到最小。

在自适应学习进行到 2s 时，网络已经能够几乎无误差地逼近期望的目标响应，误差降低到接近 0 的水平。

9.6.3 自适应滤波系统辨识

【例 9-12】 线性神经网络自适应系统辨识实例。本例中我们设计一个线性网络，该网络能够随着被辨识的模型的变化而变化，从而对实现对一个线性系统的自适应辨识。

解： ① 设定输入信号持续时间为 5s，采样率为 200Hz。输入如下命令：

```
time1 = 0:0.005:3;
time2 = 3.005:0.005:6;
time = [time1 time2];
P = sin(sin(time*4).*time*8);
```

在图上绘出输入信号的曲线。输入命令：

```
plot(time,P);
xlabel('时间');
ylabel('输入信号');
```

结果如图 9-27 所示。

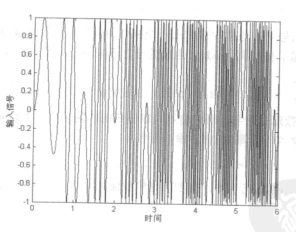


图 9-27 输入信号曲线图

② 设定系统的期望响应为 T，对于前 3s 和后 3s，响应方式是不一样的。输入如下命令：

```
steps1=length(time1);
steps2=length(time2);
[T1,state]=filter([1 -0.5],1,P(1:steps1));
T2=filter([0.9 -0.6],1,P(1+steps1:steps2+steps1),state);
```

```
T=[T1 T2];
```

在图上绘出期望输出曲线图。输入命令：

```
figure,
plot(time,T);
xlabel('时间');
ylabel('输出信号');
```

输出结果如图 9-28 所示。

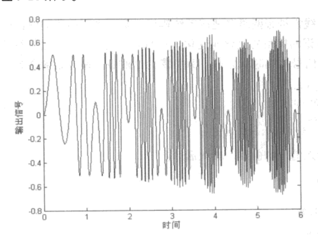


图 9-28　期望输出信号曲线图

③ 采用输入信号 P 在当前时刻和上一个采样时刻的值作为滤波器输入，利用这两点来估计输入信号的值。同样利用 adapt 函数进行自适应滤波。输入命令：

```
T=con2seq(T);
P=con2seq(P);  %将输入和输出向量转成 adapt 需要的序列信号格式
```

利用 newlin 函数建立网络，网络输入的取值范围由输入信号 P 的最小和最大值确定，两点延迟滤波，最大学习速率为 0.5。输入命令：

```
net = newlin(minmax(cat(2,P{:})), 1,[0 1],0.5);
```

④ 利用 adapt 函数进行训练，得到网络的权值和阈值。输入如下命令：

```
[net,a,e]=adapt(net, P, T);
```

绘出网络自适应滤波的输出与期望响应。输入命令：

```
figure;
plot(time, cat(2,a{:}), 'b', time, cat(2,T{:}), '.');
xlabel('时间');
legend({'输出','期望响应'});
```

得到绘图结果如图 9-29 所示。

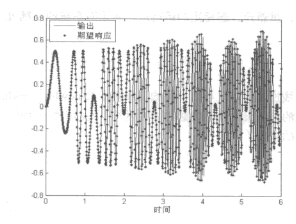


图 9-29　实际输出与期望响应信号曲线图

图中曲线连接为实际输入信号，圆点为期望响应信号，由于信号比较复杂，因此仍不清楚。下面画出误差曲线。输入命令：

```
figure,
plot(time,cat(2,e{:}));
xlabel('时间');
ylabel('误差');
```

得到的误差曲线结果如图 9-30 所示。

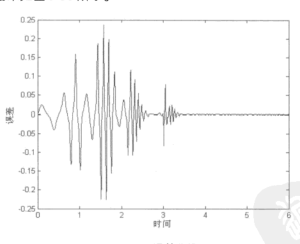


图 9-30　误差曲线图

可以看到，自适应线性滤波经过 2.5s 的学习，然后变得比较精确了，由于在第 3 秒的时候系统期望响应发生了变化，网络又用了大约 0.5s 的时间来学习与跟踪模型，从而实现了系统辨识的功能。

非线性系统处于某一个工作状态时，自适应线性系统能够很好地逼近非线性系统的状态，而一旦非线性系统发生了突变，自适应系统又能够很快地跟踪系统的变化状况，通过

一定时间的学习训练，重新建立起稳定的运行点，实现对系统的高精度逼近。

9.7 小结

本章主要介绍了线性网络以及与之相关的工具箱函数 newlin、newlind，讲解了如何针对一般的目的或特定的目的设计合适的线性神经网络，同时介绍与线性神经网络密切相关的线性自适应滤波器的应用。



第 10 章　BP 神经网络

20 世纪 80 年代，David Rumelhart、Geoffrey Hinton 以及 Williams 分别独立地给出 BP 算法的清楚的表述，解决了多层神经网络的学习问题，实现了多层网络的设想。这重新激起了人们因 Minsky 等人对单层感知器局限性的判定而失去的信心，BP 算法的出现极大地促进了神经网络的发展。

BP 神经网络又称为误差反向传播（Back Propagation）神经网络，它是一种多层的前向型神经网络。在 BP 网络中，信号是前向传播的，而误差是反向传播的。BP 网络通常具有一个或多个 sigmoid 隐层和线性输出层，能够对具有有限个不连续点的函数进行逼近。

所谓的反向传播是指误差的调整过程是从最后的输出层依次向之前各层逐渐进行的。标准的 BP 网络采用梯度下降算法，与 Widrow-Hoff 学习规则相似，网络权值沿着性能函数的梯度的反向调整。

目前在神经网络的多数应用中，都采用的是 BP 网络及其变化形式。BP 神经网络是前向型网络的核心部分，具有广泛的适应性和有效性，主要应用于模式识别与分类、数据压缩和函数逼近等方面。

10.1 BP 神经网络结构

BP 神经网络具有 sigmoid 隐层以及线性输出层，具有很强的映射能力，本节我们将对 BP 网络神经元和网络结构进行介绍。

10.1.1 BP 网络神经元模型

图 10-1 给出了一个具有 R 个输入的基本的 BP 神经元模型结构。图中每一个输入被赋予一定的权值，与偏差求和后形成神经元传递函数的输入。

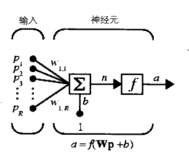


图 10-1　BP 神经元模型示意图

BP 网络属于多层网络，其神经元常用的传递函数包括 log-sigmoid 型函数 logsig、tan-sigmoid 函数 tansig，以及线性函数 purelin，如图 10-2 所示。

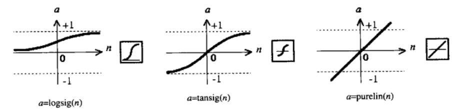


图 10-2　BP 神经元常用的几种传递函数

需要指出，sigmoid 型传递函数的曲线形状是 S 型的，log-sigmoid、tan-sigmoid 型函数都是如此。

如果 BP 网络的输出层采用 sigmoid 型传递函数，那么网络的输出就限制在[−1，+1]范围之内，而如果采用线性函数 purelin 作为输出层的传递函数，那么输出可以取任意值。因此在隐层中常常采用 sigmoid 函数进行中间结果的传递，而在最后输出层用线性传递函数对输出进行值域扩展。

在 BP 神经网络的训练过程中，计算传递函数的导数是非常重要的。对于图 10-2 中所示的任意一种传递函数，在 MATLAB 中都可以通过在函数名前加“d”而得到对应的微分函数，即 dlogsig、dtansig、dpurelin。要查询这些传递函数的导数函数名，只需在工作区对传递函数参数输入“deriv”就可以得到。

以上 3 种函数是 BP 网络中最常用到的传递函数，根据用户需要也可以在 MATLAB 中自己创建其他形式的传递函数。

10.1.2　BP 神经网络结构

图 10-3 显示了一个具有 R 个输入，由 S 个 logsig 神经元构成的单层网络。其中，左边是详细结构图，右边是以向量形式表示的结构图。

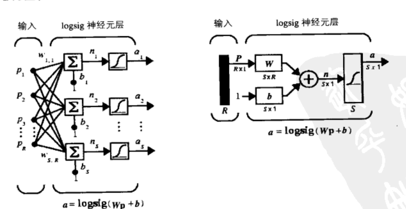


图 10-3　单层 logsig 神经元构成的网络结构

前向型神经网络通常具有一个或多个由 sigmoid 神经元构成的隐层，以及一个由线性神经元构成的输出层。多个具有非线性传递函数的神经元层使得网络可以学习输入和输出之间的非线性关系，而线性输出层使得网络可以产生[−1，+1]之外的输出值。

当然，如果恰好需要对网络输出的值域加以限制，例如想要将网络输出值限制在[0，1]以内，那么输出层就应该采用 sigmoid 类型的传递函数。

图 10-4 是一个典型的具有两个神经元层的 BP 神经网络，隐层传递函数为 tansig，输出层传递函数为 purelin 函数。

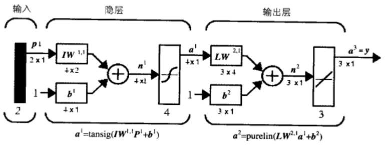


图 10-4 由两层神经元构成的 BP 网络结构

10.2 BP 网络学习规则

BP 网络的学习仍然是有监督学习，训练过程需要提供输入向量 $\boldsymbol{p}$ 和期望响应 $\boldsymbol{t}$，训练过程中网络的权值和偏差根据网络误差性能进行调整，最终实现期望的功能。前向型神经网络仍然采用均方误差作为默认的网络性能函数，网络学习的过程就是使均方误差最小化的过程。

BP 网络的学习算法有很多变化形式，对应的训练函数包括 traingd、traingdm、traingdx、trainrp、traincgf、traincgp、traincgb、trainscg、trainbfg、trainoss、trainlm、trainbr 等。各种训练函数采用不同的训练算法，其中最简单的 BP 算法仍然是最速下降法。本节将对其中最主要的几种进行介绍。

10.2.1 BP 算法

BP 算法沿着误差函数减小最快的方向，也就是梯度的反方向改变权值和偏差，这一点与线性网络的学习算法是一致的。BP 算法的迭代计算公式可以表示为：

$$x_{k+1} = x_k - a_k g_k$$

其中，x_k 代表当前的权值和偏差，x_{k+1} 代表迭代产生的下一次的权值与偏差，g_k 为当前误差函数的梯度，a_k 代表学习速率。

下面以含有两个隐层，即一共具有 4 个神经元层的 BP 网络为例，对学习算法进行推导。设输入的数目为 M，其中任意一个用 m 来标记；第 1 个隐层记为 I，包含 I 个神经元，其中任意一个神经元用 i 来标记；第 2 个隐层记为 J，包含 J 个神经元，其中任意一个神

经元用 j 来标记；输出层记为 P，包含 P 个神经元，其中任意一个用 p 来标记。

例如，输入层与第 1 隐层之间的权值记为 w_{mi}，表示从输入层第 m 个神经元输出到第 1 隐层第 i 个神经元之间的权值；第 1 隐层与第 2 隐层之间的权值记为 w_{ij}；第 2 隐层与输出层之间的权值记为 w_{jp}。

神经元的输入记为 u，输出记为 v，用上标表示神经元所处层，下标表示层中的序号，如 u_i^I 表示第 1 隐层第 i 个神经元的输入。设所有神经元的传递函数均为 sigmoid 函数；训练样本集为 $\boldsymbol{X}=[X_1, X_2, \cdots, X_N]$，其中任意一个训练样本 X_K 都是一个 M 维矢量，即 $\boldsymbol{X}_k=[X_{k1}, X_{k2}, ..., X_{kM}]$，（$k=1,2,...,N$）；期望响应为 $\boldsymbol{d}_k=[d_{k1}, d_{k2}, ..., d_{kP}]^T$，实际输出为 $\boldsymbol{Y}_k=[Y_{k1}, Y_{k2}, ..., Y_{kP}]^T$。设 n 为迭代次数，权值和实际输出都是 n 的函数。

当网络输入训练样本为 $\boldsymbol{X}_k=[X_{k1}, X_{k2}, ..., X_{kM}]$ 时，网络中信号以前向的方式传递，对于各层的中间值，可以写出表达式如下：

第 1 隐层第 i 个神经元的输入为：$u_i^I=\sum_{m=1}^{M} w_{mi} x_{km}$

第 1 隐层第 i 个神经元的输出为：$v_i^I=f\left(\sum_{m=1}^{M} w_{mi} x_{km}\right)$

第 2 隐层第 j 个神经元的输出为：$v_j^J=f\left(\sum_{i=1}^{I} w_{ij} v_i^I\right)$

输出层第 p 个神经元输入为：$u_p^P=\sum_{j=1}^{J} w_{jp} v_j^J$

输出层第 p 个神经元输出，即网络输出为：$y_{kp}=v_p^P=f\left(\sum_{j=1}^{J} w_{jp} v_j^J\right)$

输出层第 p 个神经元的输出误差为：$e_{kp}(n)=d_{kp}(n)-y_{kp}(n)$

定义误差能量为 $\frac{1}{2} e_{kp}{}^2(n)$，输出层所有神经元的误差能量总和为：$E(n)=\frac{1}{2} \sum_{p=1}^{P} e_{kp}{}^2(n)$

误差与信号相反，从后向前传播，在反向传播的过程中，逐层地修改权值和偏差。下面计算此反向传播和误差调整的过程。

1．隐层 J 与输出层 P 之间权值的调整

BP 算法中，权值的调整量与输出相对于期望响应的误差能量对权值的偏微分大小成正比，符号相反，下面计算此偏微分的值。

$$\frac{\partial E(n)}{\partial w_{jp}(n)}=\frac{\partial E(n)}{\partial e_{kp}(n)} \cdot \frac{\partial e_{kp}(n)}{\partial y_{kp}(n)} \cdot \frac{\partial y_{kp}(n)}{\partial u_p^P(n)} \cdot \frac{\partial u_p^P(n)}{\partial w_{jp}(n)} \tag{10.1}$$

由误差能量定义以及各变量之间的关系可知：

$$\frac{\partial E(n)}{\partial e_{kp}(n)}=e_{kp}(n), \quad \frac{\partial e_{kp}(n)}{\partial y_{kp}(n)}=-1, \quad \frac{\partial y_{kp}(n)}{\partial u_p^P(n)}=f'(u_p^P(n)), \quad \frac{\partial u_p^P(n)}{\partial w_{jp}(n)}=v_j^J(n) \tag{10.2}$$

从而此偏微分的值为：

$$\frac{\partial E(n)}{\partial w_{jp}(n)} = -e_{kp}(n) \cdot f'(u_p^P(n)) \cdot v_j^J(n) \tag{10.3}$$

定义局部梯度为：

$$\delta_p^P(n) = -\frac{\partial E(n)}{\partial u_p^P(n)} = e_{kp}(n) \cdot f'(u_p^P(n)) \tag{10.4}$$

根据梯度下降学习规则，$w_{jp}(n)$ 的修正量为：

$$\Delta w_{jp}(n) = -\eta \frac{\partial E(n)}{\partial w_{jp}(n)} = \delta_p^P(n) \cdot v_j^J(n) \tag{10.5}$$

其中 η 为学习步长，$\delta_p^P(n)$ 可以根据（10.4）式求得，$v_j^J(n)$ 可以根据信号正向传播过程求得，从而可以计算出下一次 $w_{jp}(n)$ 的迭代值。

例如，当最后一层采用 sigmoid 传递函数：

$$f(x) = \frac{1}{1+\exp(-ax)}, a > 0 \tag{10.6}$$

时，如果 $a=1$，可以求得：

$$\delta_p^P(n) = f'(u_p^P(n)) \cdot e_{kp}(n) = y_{kp}(n) \cdot (1 - y_{kp}(n)) \cdot (d_{kp}(n) - y_{kp}(n)) \tag{10.7}$$

将（10.7）式代入（10.5）式中，即得到隐层 J 与输出层 P 之间权值 $w_{jp}(n)$ 的下一次迭代值：

$$w_{jp}(n+1) = w_{jp}(n) + \Delta w_{jp}(n) \tag{10.8}$$

2．隐层 I 与隐层 J 之间权值的调整

与上一层权值的调整类似，I、J 隐层权值的调整也沿着梯度下降方向进行调整，其修正量为：

$$\Delta w_{ij}(n) = -\eta \frac{\partial E(n)}{\partial w_{ij}(n)} = \eta \delta_j^J(n) \cdot v_i^I(n) \tag{10.9}$$

其中，局部梯度 $\delta_j^J(n)$ 定义为：

$$\delta_j^J(n) = -\frac{\partial E(n)}{\partial u_j^J(n)} = -\frac{\partial E(n)}{\partial v_j^J(n)} \cdot \frac{\partial v_j^J(n)}{\partial u_j^J(n)} = -\frac{\partial E(n)}{\partial v_j^J(n)} \cdot f'(u_j^J(n)) \tag{10.10}$$

将误差能量的定义代入（10.10）式，经过化简后可以求得：

$$\delta_j^J(n) = f'(u_j^J(n)) \cdot \sum_{p=1}^{P} \delta_p^P(n) \cdot w_{jp}(n) \tag{10.11}$$

$\delta_p^P(n)$ 是输出层的对应的局部梯度，已经在上一步计算中求得。对于假定的 sigmoid 函数，$f'(u_j^J(n)) = v_j^J(n)(1 - v_j^J(n))$，因此代入（10.11）式就可以得到隐层 I 与隐层 J 之间权值 $w_{ij}(n)$ 的下一次迭代值：

$$w_{ij}(n+1) = w_{ij}(n) + \Delta w_{ij}(n) \tag{10.12}$$

3. 输入层 M 与隐层 I 之间权值的调整

与上面的推导类似，两层之间任意两节点之间的权值修正量为：

$$\Delta w_{mi}(n) = \eta \delta_i^I(n) \cdot x_{km}(n) \tag{10.13}$$

其中的局部梯度 $\delta_i^I(n)$ 为：

$$\delta_i^I(n) = f'(u_i^I(n)) \cdot \sum_{j=1}^{J} \delta_j^J(n) \cdot w_{ij}(n) \tag{10.14}$$

而传递函数的导数 $f'(u_i^I(n)) = v_i^I(n)(1 - v_i^I(n))$，$w_{ij}(n)$ 已经在上一层的计算中求得，因此通过下面的迭代公式，就可以求得下一步迭代的输入层 M 与隐层 I 之间权值。

$$w_{mi}(n+1) = w_{mi}(n) + \Delta w_{mi}(n) \tag{10.15}$$

以上是我们以包含两个隐层的全 sigmoid 传递函数 BP 神经网络为例，导出的 BP 学习规则的权值迭代公式，偏差的迭代公式也可以用同样的方法求得。

梯度下降算法的实现有两种方式：一种是递增模式，另一种是批处理模式。在递增模式下，网络每获得一个新的输入样本，就计算一次梯度并更新权值；而在批处理模式下，网络需要获得所有的输入样本，然后根据所有的输入样本来更新权值。下面的一节将对批处理训练模式进行介绍。

10.2.2 批处理学习算法

在批处理模式下，网络权值和偏差只在整个输入样本集都已经获取之后才进行更新。将每次样本输入时计算得到的梯度加起来，得到最终的权值和偏差。

我们知道，利用 MATLAB 工具箱函数 train 可以完成对网络的训练，在训练之前将网络对象的训练算法 trainFcn 的值指定为相应的算法即可；而另一种训练的方法是，直接调用相关训练算法的函数。

例如基本的 BP 梯度下降算法的训练可以通过调用函数 traingd 来实现，而为了稳定迭代过程，减少振荡，人们发展了带动量的梯度下降算法 traingdm 来平滑训练过程。

标准的梯度下降算法和有动量的梯度下降算法应用于实际问题时往往有学习速率过慢的缺陷，而且容易陷入局部极小点，因此人们提出了多种改进的高效 BP 算法。下面将介绍的高效算法的训练速率通常为以上两种算法的数十倍到数百倍，而这些算法都是基于批处理训练模式的。

这些快速学习算法主要分为两类。

- 第一类是启发性学习算法，包括：可变学习速率的梯度下降法、有动量和自适应学习速率的梯度下降法、弹性 BP 训练法等等。
- 第二类则是基于最优化理论的训练算法，包括共轭梯度算法、拟牛顿法、Levenberg-Marquardt 算法等等。

这些改进算法在 MATLAB 中都有各自对应的训练函数，我们将在后续章节中进行介绍。

10.3 BP 网络的 MATLAB 实现

本节我们主要通过实例介绍MATLAB中与BP神经网络相关的网络创建以及各种网络训练函数的应用。

10.3.1 BP 网络的创建与仿真

MATLAB 神经网络工具箱提供函数 newff 来创建一个前向型 BP 神经网络。其常用的调用格式为：

net = newff(***P***, ***T***, [S1 S2...S(N-l)], {TF1 TF2...TFNl})

其中：

- ***P***，***T*** 分别为输入样本和期望响应；
- Si 为网络各层的神经元数目；
- TFi 为网络各层的传递函数类型。

默认情况下，网络中间隐层采用 tansig 函数，输出层采用 purelin 函数。函数执行的结果是返回一个 N 层前向型的 BP 神经网络。

利用 sim 函数仍然可以实现网络对一定输入的仿真，得到其仿真输出。

【例 10-1】 BP 网络创建实例。给定一定的输入样本和目标响应，应用 newff 函数生成一个单输入的单隐层 BP 网络并对其进行仿真。

解： ① 首先定义网络的输入向量与期望响应。输入命令：

```
P = [0 1 2 3 4 5 6 7 8 9 10];
T = [0 1 2 3 4 3 2 1 2 3 4];
```

② 然后调用 newff 函数生成一个单隐层的 BP 网络，该隐层包含 5 个神经元。输入命令：

```
net = newff(P,T,5);
```

③ 对网络进行仿真，并将结果以图形输出。输入命令：

```
Y = sim(net,P);
plot(P,T,P,Y,'o');
```

绘出的图形如图 10-5 所示。

在图 10-5 中直线代表期望响应，圆点代表真实的网络响应，两者有一定的差距。此外要注意的是，newff 函数直接生成的网络具有一定随机性，重复上述命令绘出的响应图每次都不一样。要使得输出接近于期望响应，还需要对网络进行训练，我们在下一节中介绍。

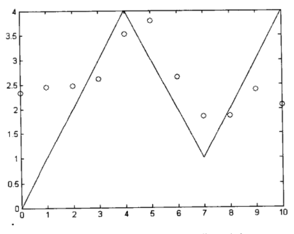


图 10-5 初始生成网络的输出与期望响应图

10.3.2 BP 网络的训练

前面我们说过，对网络进行训练既可以在指定学习算法的条件下调用普通的 train 函数，也可以直接调用不同训练算法的对应函数。要应用不同的训练算法对网络进行训练，可以执行下面两个步骤：

（1）设置生成的网络对象.trainParam.Fcn 为对应的函数名；

（2）调用 train 函数对网络进行训练。

下面对其进行实例说明。

1．直接用 train 函数对 BP 网络进行训练

【例 10-2】 BP 网络训练实例 1。针对【例 10-1】中的网络，用 train 函数对其进行训练并仿真，使其适应于给定的输入和输出样本。

解： 这里直接调用 train 函数来对网络进行训练，设定训练的目标为 0.001，并绘出输出图与期望响应进行比较。

在完成【例 10-1】的基础上，继续输入如下命令：

```
net.trainParam.Fcn = 'traingd';
net.trainParam.goal = 0.001;
net = train(net,P,T);
Y = sim(net,P);
plot(P,T,P,Y,'o');
```

注意因为创建网络时，默认的网络学习函数为'trainlm'，因此这里网络训练采用的学习算法是'traingd'，经过训练后，绘出的输出图形如图 10-6 所示。

可以看到，训练后的网络输出与期望响应符合程度比训练前有了很大的提高。

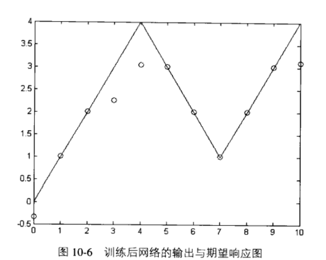


图 10-6 训练后网络的输出与期望响应图

2. 利用标准的梯度下降算法对网络进行训练

标准的梯度下降算法对应算法名为“traingd”。因此，用基本的 BP 训练算法对网络进行训练，只需要将生成网络的 net.trainParam.Fcn 参数设置成'traingd'，然后再调用 train 函数即可。下面举例说明。

【例 10-3】 BP 网络训练实例 2。对样本数据 P=[−1，−1，2，2；0，5，0，5]，T=[−1，−1，1，1]，生成 BP 网络并应用基本的梯度下降算法对其进行训练。

解： ① 首先定义输入和输出样本。输入命令：

```
P=[-1 -1 2 2; 0 5 0 5];
T=[-1 -1 2 2];
```

② 然后生成与输入、输出对应的 BP 网络。输入命令：

```
net=newff(minmax(P),[3,1],{ 'tansig','purelin'},'traingd');
```

③ 接下来设定网络训练参数。输入如下命令：

```
net.trainParam.lr=0.05;
net.trainParam.epochs=300;
net.trainParam.goal=1e-5;
```

④ 下面对网络进行训练。输入命令：

```
[net,tr]=train(net,P,T);
```

在弹出的窗口中单击“Performance”按钮，训练过程中网络误差性能的变化如图 10-7 所示。

⑤ 经过 158 次迭代，训练后网络的误差降到了 1e-5 以下，训练完成。我们下面对网络进行仿真。输入命令：

```
Y = sim(net,P),
```

输出结果为：

```
Y =   -0.9969   -1.0003    2.0043    1.9966
```

可以看到，网络的输出与期望响应是很接近的，这表明经过训练后的 BP 网络完成了输入、输出样本的映射。

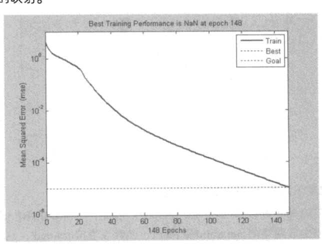


图 10-7　训练中网络误差性能的变化图

3. 有动量的梯度下降算法

标准的梯度下降法在调整权值时，仅仅按照当前时刻的负梯度方向进行调整，并没有考虑以前各次运算步骤中的梯度方向，因此新的样本对迭代过程影响太大，可能会导致训练过程中调整方向发生振荡，导致不稳定和收敛速度慢的问题。有动量的梯度下降算法则考虑了往前时刻的贡献，其权值迭代公式如下所示：

$$w_{ij}(n+1)=w_{ij}(n)+\eta[(1-\alpha)D(n)+\alpha D(n-1)]$$

式中，$D(n)$，$D(n-1)$ 分别表示 n 时刻，n-1 时刻的负梯度。由于加入了以前时刻梯度的贡献，相当于给迭代过程添加了一个低通滤波器，这样可以使得网络忽略误差曲面上的细节特征，从而避免了陷入局部极小点的问题。

在 MATLAB 中，有动量的梯度下降算法对应的函数为 traingdm，下面举例说明应用 traingdm 的训练过程。

【例 10-4】 BP 网络训练实例 3。对【例 10-3】中的输入和输出样本，采用带动量的梯度下降算法对生成网络进行训练。

解： ① 利用 newff 函数生成网络。在 newff 函数中，设定学习算法为 traingdm。输入如下命令：

```
P=[-1 -1 2 2; 0 5 0 5],
T=[-1 -1 2 2];
net=newff(minmax(P),[3,1],{ 'tansig','purelin'},'traingdm');
```

② 同样，设定网络训练参数并进行训练。输入命令如下：

```
net.trainParam.lr=0.05;
net.trainParam.epochs=300;
net.trainParam.goal=1e-5;
[net,tr]=train(net,P,T);
```

在弹出的窗口中单击“Performance”按钮，生成的误差性能曲线图如图 10-8 所示。

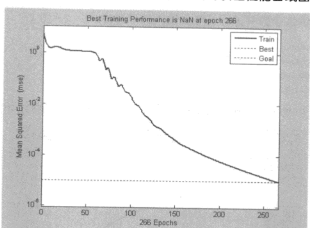


图 10-8 有动量的梯度下降算法训练过程中网络误差性能的变化图

③ 可以看到，经过了 266 次迭代之后，网络训练达到了期望的误差水平。

下面应用 sim 函数进行网络仿真。输入命令：

```
Y = sim(net,P),
```

输出结果为：

```
Y =   -1.0031    -0.9982    2.0037    1.9964
```

这说明，训练后的网络是能够完成输入与输出样本之间的映射的。

4．可变学习速率的梯度下降算法

在标准的梯度下降算法中，训练过程中学习速率是固定的。算法的性能对于学习速率的设定值非常敏感。如果学习速率设得过高，可能导致训练过程出现振荡以及不稳定；而如果学习速率设得过低，又会导致收敛时间过长。然而，事先知道一个最佳的学习速率并不是一件很容易的事，并且随着训练的进行，网络的误差性能点在误差性能曲面上不断地移动，最佳的学习速率也是随之不断变动的。

如果将学习速率设定为可变的，那么就可以改善训练算法的性能。具备自适应学习速率的梯度下降算法可以在保持训练过程稳定的前提下采用最大的学习步长。当前的学习速率是根据误差曲面的局部复杂性来确定的。

在 MATLAB 中，可变学习速率的梯度下降算法对应的函数为 traingda。下面对其应用效果进行举例说明。

【例 10-5】 BP 网络训练实例 4。对【例 10-4】中的输入和输出样本，采用可变学习速率梯度下降算法对生成网络进行训练。

解：① 利用 newff 函数生成网络。在 newff 函数中，设定学习算法为 traingda。输入命令：

```
P=[-1 -1 2 2; 0 5 0 5];
T=[-1 -1 2 2];
net=newff(minmax(P),[3,1],{ 'tansig','purelin'},'traingda');
```

② 设定网络训练参数并进行训练。输入如下命令：

```
net.trainParam.lr=0.05;
net.trainParam.epochs=300;
net.trainParam.goal=1e-5;
[net,tr]=train(net,P,T);
```

在弹出的窗口中单击"Performance"按钮，生成的误差性能曲线如图 10-9 所示。

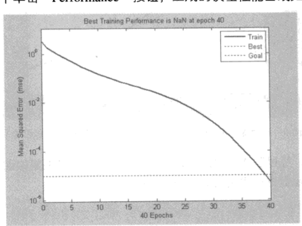


图 10-9 可变学习速率的梯度下降算法训练过程中网络误差性能的变化

可以看到，经过了 40 次迭代，网络就达到了期望的误差性能，这相对于前面的 traingd 和 traingdm 的训练来说，训练速度得到了很大的提升。

③ 接下来应用 sim 函数进行仿真。输入命令：

```
Y = sim(net,P),
```

输出结果为：

```
Y =   -1.0001   -0.9972    1.9966    2.0009
```

这说明以 traingda 算法训练后的网络同样是符合要求的。

5. 弹性梯度下降算法

多层神经网络通常在隐层中采用 sigmoid 型的传递函数。由于 sigmoid 函数的作用是将无限的输入范围压缩到一个有限的输出范围内，因此它们都有这样一个特性，即输入样本变得非常大的时候，函数曲线斜率将会逐渐接近于 0。这可能造成的一个问题，即如果采用最陡下降法对网络进行训练，梯度数值有可能会很小，这样每一次迭代权值和偏差的

改变量会很小，尽管它们距离最优值还有很远的距离。

有弹性的 BP 训练算法的目标就是消除由于误差函数偏微分的数值上变动所造成的不利于训练的效应。在这种算法下，只通过偏微分函数符号决定权值的变化方向，而忽略偏微分数值的大小，权值变化量则由一个独自更新的数值来决定。

如果连续两次迭代中误差性能函数对某一权值的偏导数的正负号相同，则权值更新值会增大；而如果连续两次迭代中误差性能函数对某一权值的偏导数的正负号相同，则权值更新值会减小，如果偏导数等于 0，则停止迭代。这样权值的变动过程是一个幅度逐渐减小的振荡过程，最后收敛到梯度为 0 的目标点。

在 MATLAB 中，弹性梯度下降算法对应的函数为 trainrp。下面我们以例题的形式对其进行说明。

【例 10-6】 BP 网络训练实例 5。对【例 10-5】的输入和输出样本，利用弹性梯度下降算法对生成网络进行训练。

解： ① 利用 newff 函数生成网络。在 newff 函数中，设定学习算法为 trainrp。输入命令：

```
P=[-1 -1 2 2; 0 5 0 5],
T=[-1 -1 2 2];
net=newff(minmax(P),[3,1],{ 'tansig','purelin'},'trainrp');
```

② 设定网络训练参数并进行训练。输入如下命令：

```
net.trainParam.lr=0.05;
net.trainParam.epochs=300;
net.trainParam.goal=1e-5;
[net,tr]=train(net,P,T);
```

在弹出的窗口中单击“Performance”按钮，误差性能曲线图如图 10-10 所示。

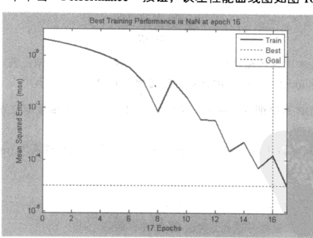


图 10-10 弹性梯度下降算法训练过程中网络误差性能的变化

可以看到，经过了 17 次迭代网络达到了期望的误差性能，并且在弹性梯度训练算法下网络误差性能的振荡变化特性表现得很明显。

③ 下面我们对其进行仿真。输入命令：

```
Y = sim(net,P),
```

输出结果为：

```
Y =   -0.9906   -0.9949    1.9825    1.9858
```

这验证了网络实现了输入和输出目标样本的映射。

6. Fletcher-Reeves 共轭梯度算法

尽管标准的 BP 算法采用梯度下降算法，权值和偏差沿误差函数下降最快的方向进行调整，但却并不一定是收敛最快的算法。在改进的 BP 训练算法中，有一大类的算法称为共轭梯度算法。在这一类算法中，权值和偏差沿着共轭梯度方向进行调整，通常能够获得比标准的梯度算法更快的收敛速度。

共轭梯度算法的第一次迭代都从最陡下降的梯度方向开始。梯度向量为：

$$\boldsymbol{p}_0 = -\boldsymbol{g}_0$$

沿着此方向进行权值和偏差的调整，公式为：

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$$

下一次搜索方向则由前两次搜索方向的共轭方向决定，表达式为：

$$\boldsymbol{p}_k = -\boldsymbol{g}_k + \beta_k \boldsymbol{p}_{k-1}$$

对于系数 β_k 的不同计算方法产生出不同的共轭梯度算法。其中 Fletcher-Reeves 方法采取的系数确定方法为：

$$\beta_k = \frac{\boldsymbol{g}_k^T \boldsymbol{g}_k}{\boldsymbol{g}_{k-1}^T \boldsymbol{g}_{k-1}}$$

此系数含义是本次迭代梯度相对于上一次迭代梯度的归一化值。在 MATLAB 中此算法对应的函数为 traincgf。下面应用此函数对【例 10-6】进行分析。

【例 10-7】 BP 网络训练实例 6。用 Fletcher-Reeves 共轭梯度算法对【例 10-6】的网络进行训练。

解： ① 利用 newff 函数生成网络。在 newff 函数中，设定学习算法为 traincgf。输入命令：

```
P=[-1 -1 2 2; 0 5 0 5],
T=[-1 -1 2 2];
net=newff(minmax(P),[3,1],{ 'tansig','purelin'},'traincgf');
```

② 设定网络训练参数并进行训练。输入如下命令：

```
net.trainParam.lr=0.05;
net.trainParam.epochs=300;
net.trainParam.goal=1e-5;
[net,tr]=train(net,P,T);
```

在弹出的窗口中，单击“Performance”按钮，生成的误差性能曲线如图 10-11 所示。

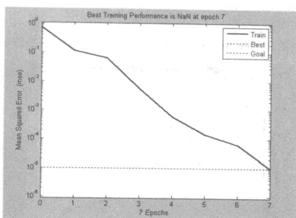


图 10-11 Fletcher-Reeves 共轭梯度算法训练过程中网络误差性能的变化

可以看到，只经过了 7 次迭代，网络就达到了期望的误差性能。这说明 Fletcher-Reeves 共轭梯度算法的收敛速度是非常快的。

③ 下面对其进行仿真。输入命令：

```
Y = sim(net,P),
```

输出结果为：

```
Y =   -0.9985   -1.0003    1.9940    2.0006
```

这个结果验证了经过训练后的 BP 网络性能符合需求。

7. Polak-Ribiére 共轭梯度算法

Polak-Ribiére 共轭梯度算法采取的共轭梯度系数计算公式如下：

$$\beta_k = \frac{\Delta \boldsymbol{g}_{k-1}^T \boldsymbol{g}_k}{\boldsymbol{g}_{k-1}^T \boldsymbol{g}_{k-1}}$$

此系数为上次迭代梯度与本次迭代的梯度的内积对本次梯度的归一化值。在 MATLAB 中此算法对应的函数为 traincgp，下面应用此算法对网络进行训练和仿真。

【例 10-8】 BP 网络训练实例 7。用 Polak-Ribiére 共轭梯度算法对**【例 10-7】**的网络进行训练。

解： ① 利用 newff 函数生成网络。在 newff 函数中，设定学习算法为 traincgp。输入命令：

```
P=[-1 -1 2 2; 0 5 0 5],
T=[-1 -1 2 2];
net=newff(minmax(P),[3,1],{ 'tansig','purelin'},'traincgp');
```

② 设定网络训练参数并进行训练。输入如下命令：

```
net.trainParam.lr=0.05;
```

```
net.trainParam.epochs=300;
net.trainParam.goal=1e-5;
[net,tr]=train(net,P,T);
```

在弹出的窗口中单击“Performance”按钮，生成的误差性能曲线如图 10-12 所示。

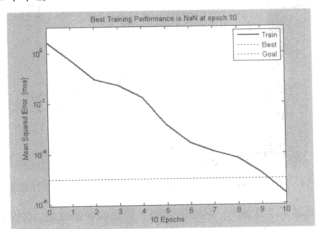


图 10-12　Polak-Ribiére 共轭梯度算法训练过程中网络误差性能的变化

可以看到，经过了 10 次迭代，网络就达到了期望的误差性能，收敛速度与 Fletcher-Reeves 共轭梯度算法基本相当。

③ 下面对其进行仿真。输入命令：

```
Y = sim(net,P),
```

输出结果为：

```
Y =   -0.9988   -1.0012    2.0027    1.9997
```

这验证了经过训练后的 BP 网络性能符合需求。

8. Powell-Beale 重置共轭梯度算法

对于所有的共轭梯度算法，搜索方向需要周期性地重置为当前负梯度方向。重置点通常选取在迭代次数与网络设置的训练参数相等的时候，但也存在其他的重置方法，以提高训练的效率，Powell-Beale 方法就是其中的一种。

在此方法中，如果上一次的梯度方向与本次梯度方向之间正交性很低的时候，就重置搜索方向。具体判据由下式决定：

$$\left|\boldsymbol{g}_{k-1}^{T}\boldsymbol{g}_{k}\right| \geqslant 0.2\|\boldsymbol{g}_{k}\|^{2}$$

一旦当前梯度与上一次梯度满足此不等式，就对搜索方向进行重置。在 MATLAB 中此算法对应的函数为 traincgb。下面应用此算法对网络进行训练和仿真。

【例 10-9】 BP 网络训练实例 8。用 Powell-Beale 共轭梯度算法对**【例 10-8】**的网络进行训练。

解：① 利用 newff 函数生成网络。在 newff 函数中，设定学习算法为 traincgb。输入命令：

```
P=[-1 -1 2 2; 0 5 0 5],
T=[-1 -1 2 2];
net=newff(minmax(P),[3,1],{ 'tansig','purelin'},'traincgb');
```

② 设定网络训练参数并进行训练。输入如下命令：

```
net.trainParam.lr=0.05;
net.trainParam.epochs=300;
net.trainParam.goal=1e-5;
[net,tr]=train(net,P,T);
```

在弹出的窗口中单击“Performance”按钮，生成的误差性能曲线如图 10-13 所示。

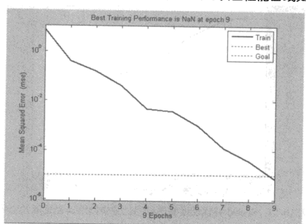


图 10-13 Powell-Beale 共轭梯度算法训练过程中网络误差性能的变化

可以看到，经过了 9 次迭代，网络达到了期望的误差性能。

③ 下面对其进行仿真。输入命令：

```
Y = sim(net,P),
```

输出结果为：

```
Y =   -1.0010   -0.9990    2.0051    2.0002
```

这验证了经过训练后的 BP 网络性能符合需求。

9. Scaled 共轭梯度算法

到目前为止，讨论过的所有共轭梯度算法都需要在每一步迭代过程中对搜索方向进行计算，这样的计算量是比较大的。对此 Moller 提出了 Scaled 梯度搜索算法，在每一步迭代过程中不计算搜索方向，以减少训练过程中的计算量。在 MATLAB 中对应此算法的函数为 trainscg，下面举例对其说明。

【例 10-10】 BP 网络训练实例 9。用 Scaled 共轭梯度算法对上例的网络进行训练。

解：① 利用 newff 函数生成网络。在 newff 函数中，设定学习算法为 trainscg。输入命令：

```
P=[-1 -1 2 2; 0 5 0 5],
T=[-1 -1 2 2];
net=newff(minmax(P),[3,1],{ 'tansig','purelin'},'trainscg');
```

② 设定网络训练参数并进行训练。输入命令如下：

```
net.trainParam.lr=0.05;
net.trainParam.epochs=300;
net.trainParam.goal=1e-5;
[net,tr]=train(net,P,T);
```

在弹出的窗口中单击“Performance”按钮，生成的误差性能曲线如图 10-14 所示。

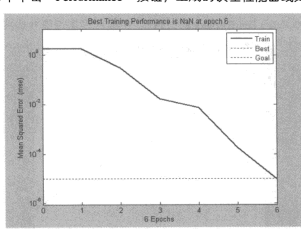


图 10-14 Scaled 共轭梯度算法训练过程中网络误差性能的变化

可以看到，经过了 6 次迭代，网络达到了期望的误差性能。

③ 下面对其进行仿真。输入命令：

```
Y = sim(net,P),
```

输出结果为：

```
Y =   -0.9970   -1.0043    1.9969    1.9994
```

这验证了经过训练后的 BP 网络性能符合需求。Scaled 共轭梯度算法收敛所需的迭代次数有时比其他共轭梯度算法要多，但是每一次迭代所需要计算量却大大减少了，因为在这种算法中，不需要计算搜索方向。Scaled 共轭梯度算法所需的存储量与 Fletcher-Reeves 方法大致相等。

10. BFGS 拟牛顿法

拟牛顿法是共轭梯度算法之外的另一种快速学习算法。标准的牛顿法迭代公式为，

$$x_{k+1} = x_k - A_k^{-1} g_k$$

其中 A_k 是误差性能函数对当前权值和偏差的二阶微分构成的 Hessian 矩阵。拟牛顿法通常比共轭梯度算法要收敛得更快，然而遗憾的是，对于前向型 BP 网络，计算 Hessian

矩阵是一项计算量耗费很大的工作。

于是人们发展了一类基于牛顿法但是不需要计算 Hessian 矩阵的算法，这一类算法被称为拟牛顿法。在这一类方法中，每一次迭代计算一个近似 Hessian 矩阵，此更新值是梯度的函数。在公开发表的拟牛顿法中，以 Broyden、Fletcher、Goldefarb 和 Shanno 发表的方法最为成功，这种方法被称为 BFGS 拟牛顿法。

MATLAB 工具箱提供了函数 trainbfg 以实现 BFGS 拟牛顿算法。下面通过实例对其进行说明。

【例 10-11】 BP 网络训练实例 10。用 BFGS 拟牛顿算法对上例的网络进行训练。

解： ① 利用 newff 函数生成网络。在 newff 函数中，设定学习算法为 trainbfg。输入命令：

```
P=[-1 -1 2 2; 0 5 0 5],
T=[-1 -1 2 2];
net=newff(minmax(P),[3,1],{ 'tansig','purelin'},'trainbfg');
```

② 设定网络训练参数并进行训练。输入如下命令：

```
net.trainParam.lr=0.05;
net.trainParam.epochs=300;
net.trainParam.goal=1e-5;
[net,tr]=train(net,P,T);
```

在弹出的窗口中单击“Performance”按钮，生成的误差性能曲线如图 10-15 所示。

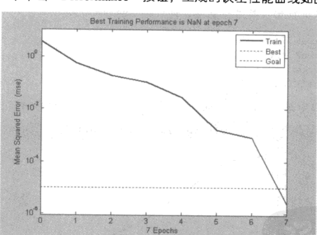


图 10-15 BFGS 拟牛顿算法训练过程中网络误差性能的变化

可以看到，经过了 7 次迭代，网络达到了期望的误差性能。

③ 下面对其进行仿真。输入命令：

```
Y = sim(net,P),
```

输出结果为：

```
Y =   -0.9969   -1.0000    1.9999    1.9996
```

这个结果验证了经过训练后的 BP 网络性能符合需求。BFGS 算法相对于共轭梯度法而言，收敛的迭代次数更少，但是通常需要更多的存储空间和计算量，这是因为需要存储近似 Hessian 矩阵的缘故。对于非常大的网络，Hessian 矩阵所消耗的存储资源是相当大的，而对于比较小的网络，BFGS 方法是一个比较有效的训练算法。

11．一步正割算法

由于 BFGS 比共轭梯度算法需要更多的存储空间和计算量，所以人们开始寻求需要更少的存储和计算资源的正割方法。一步正割算法就是介于拟牛顿法与共轭梯度算法之间的算法，它不需要存储整个 Hessian 矩阵，并且假定对于每一次迭代，上一次的 Hessian 矩阵都是恒定不变的。这样带来的好处是，在计算新的搜索方向时就不需要对矩阵求逆了。

MATLAB 工具箱提供了函数 trainoss 以实现一步正割算法。下面通过实例对其进行说明。

【例 10-12】 BP 网络训练实例 11。用一步正割算法对上例的网络进行训练。

解： ① 利用 newff 函数生成网络。在 newff 函数中，设定学习算法为 trainoss。输入命令：

```
P=[-1 -1 2 2; 0 5 0 5],
T=[-1 -1 2 2];
net=newff(minmax(P),[3,1],{ 'tansig','purelin'},'trainoss');
```

② 设定网络训练参数并进行训练。输入如下命令：

```
net.trainParam.lr=0.05;
net.trainParam.epochs=300;
net.trainParam.goal=1e-5;
[net,tr]=train(net,P,T);
```

在弹出的窗口中单击“Performance”按钮，生成的误差性能曲线如图 10-16 所示。

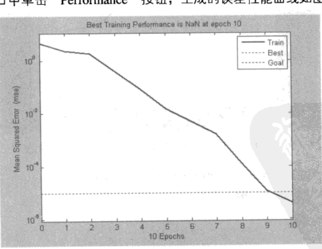


图 10-16　一步正割算法训练过程中网络误差性能的变化

可以看到，经过了 10 次迭代，网络达到了期望的误差性能。

③ 下面对其进行仿真。输入命令：

```
Y = sim(net,P),
```

输出结果为：

```
Y =   -1.0022   -0.9974   2.0020   2.0007
```

这个结果验证了经过训练后的 BP 网络性能符合需求。一步正割算法相对于 BFGS 拟牛顿法而言，需要的计算量和存储资源都有所减少，但是相对于共轭梯度算法而言有所增加，可以认为是拟牛顿法与共轭梯度算法的折中。

12. Levenberg-Marquardt 算法

和拟牛顿算法一样，Levenberg-Marquardt 算法也是期望在不计算 Hessian 矩阵的情况下获得高阶的训练速度。我们知道，误差性能函数可以表示为平方和的形式，此时，Hessian 矩阵可以近似为：

$$\boldsymbol{H} = \boldsymbol{J}^T\boldsymbol{J}$$

而梯度为：

$$\boldsymbol{g} = \boldsymbol{J}^T\boldsymbol{e}$$

其中，$\boldsymbol{J}^T$ 为雅可比矩阵，包含了网络误差函数对于权值和偏差的一阶导数，$\boldsymbol{e}$ 是网络误差向量。雅可比矩阵可以通过标准的 BP 算法计算得到，这样的计算量要比直接计算 Hessian 矩阵减少很多。

Levenberg-Marquardt 算法的更新过程与牛顿法类似，其中应用了上述方法来近似计算 Hessian 矩阵。公式表达如下：

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - [\boldsymbol{J}^T\boldsymbol{J} + \mu\boldsymbol{I}]^{-1}\boldsymbol{J}^T\boldsymbol{e}$$

其中，如果标量因子 $\mu=0$ 的话，就变成近似 Hessian 矩阵的拟牛顿法；如果因子 μ 很大的话，即成为小步长的梯度下降法，由于牛顿法在误差极小点附近通常能够确定收敛得更快更精确，因此算法的目的就是尽快转换为牛顿法。因此如果训练成功，误差性能函数减小，那么就减小 μ 的值；而如果训练失败，就增加 μ 的值。利用这样的方法，可以使得误差性能函数随着迭代的进行而下降到极小值。

MATLAB 工具箱提供了函数 trainlm 以实现 Levenberg-Marquardt 算法。我们利用 newff 函数生成神经网络的时候，如果不指定学习算法，网络的默认学习算法就是 Levenberg-Marquardt。如【例 10-2】直接调用 train 函数进行训练即可。下面再通过另一个实例对其应用进行说明。

【例 10-13】 BP 网络训练实例 12。用 Levenberg-Marquardt 算法对上例的网络进行训练。

解： ① 利用 newff 函数生成网络。在 newff 函数中，设定学习算法为 trainlm。输入命令：

```
P=[-1 -1 2 2; 0 5 0 5] ;
T=[-1 -1 2 2];
net=newff(minmax(P),[3,1],{ 'tansig','purelin'},'trainlm');
```

② 设定网络训练参数并进行训练。输入如下命令：

```
net.trainParam.lr=0.05;
net.trainParam.epochs=300;
net.trainParam.goal=1e-5;
[net,tr]=train(net,P,T);
```

在弹出的窗口中单击"Performance"按钮，生成的误差性能曲线如图 10-17 所示。

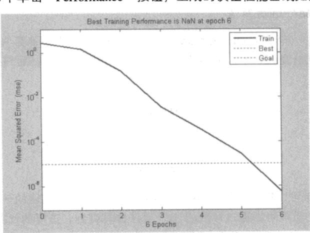


图 10-17 Levenberg-Marquardt 算法训练过程中网络误差性能的变化

可以看到，经过了 6 次迭代，网络达到了期望的误差性能。

③ 下面对其进行仿真。输入命令：

```
Y = sim(net,P),
```

输出结果为：

```
Y =   -0.9991   -0.9996    2.0011    2.0000
```

这个结果验证了经过训练后的 BP 网络性能符合需求。对于包含几百个权值的中等规模的前向网络，Levenberg-Marquardt 算法是目前最快的训练算法，因此，用 newff 函数生成的网络采用此算法作为默认训练函数。

表 10-1 列出了以上介绍的主要训练算法以及 MATLAB 对应的函数。

表 10-1 主要的 BP 网络训练算法及其 MATLAB 函数总结

训练算法缩写	MATLAB 函数	说　明
GD	traingd	标准梯度下降算法
GDM	traingdm	带动量的梯度下降算法

续表

训练算法缩写	MATLAB 函数	说　明
GDA	traingda	可变学习速率梯度算法
LM	trainlm	Levenberg-Marquardt 算法
BFG	trainbfg	BFGS 拟牛顿法
RP	trainrp	弹性梯度算法
SCG	trainscg	Scaled 共轭梯度算法
CGB	traincgb	Powell-Beale 共轭梯度算法
CGF	traincgf	Fletcher-Powell 共轭梯度算法
CGP	traincgp	Polak-Ribiére 共轭梯度算法
OSS	trainoss	一步正割算法

10.4 BP 网络的局限性

标准的梯度下降算法的训练速度通常是非常慢的，为了保证训练过程的稳定性，标准的梯度算法需要低学习速率；而有动量的学习算法相对来说速度就要快一些，因为它能够在保持训练过程的稳定性的前提下，选择更快的学习速率。这两种方法通常只用在递增模式的训练过程中。

对于中小规模的 BP 神经网络，最好采用 Levenberg-Marquardt 算法，它具有最快的收敛速度和较低的存储量，但是如果存储空间很紧缺的话，需要选择其他类型的快速算法。对于大型的网络，最好选择 Scaled 共轭梯度算法或者弹性梯度算法。

多层神经网络克服了单层感知器和线性网络的缺陷，可以对任何线性或非线性的输入、输出关系实现逼近。然而，尽管经过训练的网络理论上可以正确地完成任何问题的求解，但是 BP 训练算法却有可能无法找到切合问题的解，训练有可能会陷入局部极小点。

对于一个非线性网络而言，选择一个合适的学习速率是一项有挑战性的工作，过高的学习速率会导致学习过程的不稳定。相反，学习速率过低有会导致训练过程消耗过长的时间。相对而言，线性网络学习速率的选取就显得更加容易。不过采用快速学习算法，默认的学习速率值的表现还是较好的。

此外，非线性网络的误差性能表面要比线性网络的更复杂。采用非线性传递函数的多层网络的误差性能表面上会产生多个局部极小点。采用梯度下降算法进行训练时，解可能会陷入局部极小点，这与训练初始点的选取也有关。因为这样，所以在发现 BP 网络对于确定的问题通过训练得到的不是正确解的时候，需要重新初始化网络，以保证最终能够得到最优解。

BP 网络对于隐层中的神经元个数敏感。隐层中神经元数目太少，可能会导致训练的不适性，而神经元数目太多，有可能导致过适性，也就是说每一个点都符合得很好，但是拟合曲线却在各个点之间振荡。

10.5 BP 神经网络设计实例

由于在隐层中采用了非线性传递函数，因此 BP 神经网络对于线性问题以外的许多问题也可以很好地解决，具有相当广泛的应用。下面对其用实例进行介绍。

10.5.1 函数逼近

BP 网络采用非线性传递函数，相对线性网络具有更强的拟合能力，因此对于现实中遇到的各种复杂的函数波形可以进行很好的逼近。

【例 10-14】 BP 网络函数逼近实例。对一个具有一定采样率的正弦函数用 BP 网络来逼近其波形。

解：① 首先定义正弦函数，采样率为 20Hz，频率为 1Hz。输入如下语句：

```
k = 1;          %设定正弦信号频率
p = [0:0.05:4]; t = sin(k*pi*p);
plot(p, t, '-'),
xlabel('时间'); ylabel('输入信号');
```

绘出的曲线如图 10-18 所示。

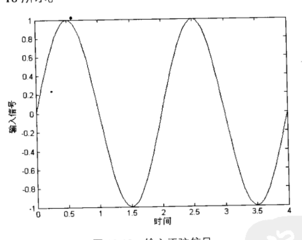


图 10-18 输入正弦信号

② 用 newff 函数生成前向型 BP 网络，设定隐层中神经元数目为 8，分别选择隐层的传递函数为 tansig，输出层的传递函数为 purelin，学习算法为 trainlm。输入如下语句：

```
net = newff(minmax(p),[10,1],{'tansig','purelin'},'trainlm');
```

对生成的网络进行仿真并作图显示。输入如下命令：

```
y1 = sim(net,p);
plot(p, t, '-', p, y1, '--');
```

绘制的曲线如图 10-19 所示。

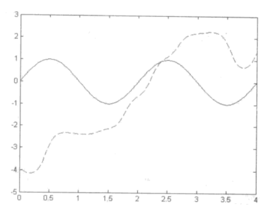


图 10-19 生成网络的仿真响应

可以看到，逼近效果相差是很远的。

③ 下面对网络进行训练，设定训练误差目标为 1e-5，最大迭代次数为 300，学习速率为 0.05。输入如下命令：

```
net.trainParam.lr=0.05;
net.trainParam.epochs=300;
net.trainParam.goal=1e-5;
[net,tr]=train(net,p,t);
```

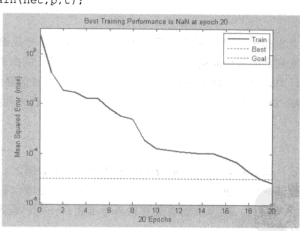


图 10-20 训练过程中网络误差性能的变化

可以看到，网络训练经过了 20 次迭代，就达到了设定的目标要求。

④ 下面对训练后的网络进行仿真，并绘图显示。输入如下命令：

```
y2 = sim(net,p);
plot(p, t, '-o', p, y2, '--');
```

绘制的曲线如图 10-21 所示。

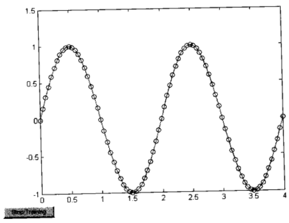


图 10-21　训练后的网络响应

可以看到，经过训练后的网络的响应与输入输出样本吻合得非常好。这也说明 BP 神经网络对于非线性函数的逼近能力是相当好的。

10.5.2　回归分析

在生产和科学实验中，选取的数学模型主要是线性回归方程形式，采用回归分析方法确定模型的参数。由于 BP 网络可对任意形状的函数曲线进行逼近，因此，可以采用 BP 网络进行回归分析。

【例 10-15】　此例是一个神经网络应用于医学中的例子，目标是设计一台仪器，能够利用光谱分析的数据测定血清中胆固醇含量。现共有 264 个病人的血清样本，在 21 个波长上进行光谱测量，同时对这些病人进行 HDL、LDL、VLDL 胆固醇含量的检测。

解： ① 在 MATLAB 神经网络工具箱中包含有这些样本数据，首先利用命令 load 对其进行加载。输入命令：

```
load choles_all;
```

数据加载完毕后，输入和输出向量 $\boldsymbol{p}$，$\boldsymbol{t}$ 出现在 MATLAB 数据区。

然后，利用函数 newff 创建网络。设定网络输入为 $\boldsymbol{p}$，$\boldsymbol{t}$，生成含有一个隐层的 BP 网络，隐层中包含 5 个神经元；定义网络首先对输入标准差归一化，设定最大比例为 0.001 以实现主成分分析。输入命令：

```
net = newff(p,t,5);
net.inputs{1}.processFcns = {'mapstd','processpca'};
net.inputs{1}.processParams{2}.maxfrac = 0.001;
net.outputs{2}.processFcns = {'mapstd'};
```

② 接下来，利用 train 函数对网络进行训练。输入如下命令：

```
[net,tr]=train(net,p,t);
```

利用 plotperform 命令可以对训练过程中网络误差的变化进行绘图显示。输入命令：

```
plotperform(tr);
```

绘出图形如图 10-22 所示。

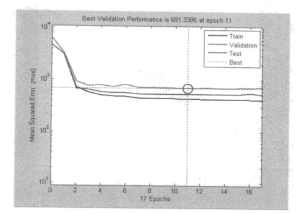


图 10-22 训练过程中网络误差性能的变化

可以看到，在 11 次迭代之后，网络的误差性能达到了期望值，因此训练终止。

③ 下面利用训练过的网络对输入的数据样本进行仿真，然后将网络输出和相应的期望输出向量进行线性回归分析。因为有三组输出值，因此需要进行三次线性回归，分别对应 hdl、ldl、vldl 线性回归。输入如下命令：

```
y = sim(net,p);
plotregression(t(1,:),y(1,:))
plotregression(t(2,:),y(2,:))
plotregression(t(3,:),y(3,:))
```

绘出的三个图形分别如图 10-23、10-24、10-25 所示。

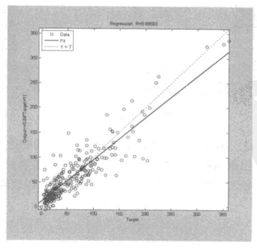


图 10-23 hdl 线性回归结果

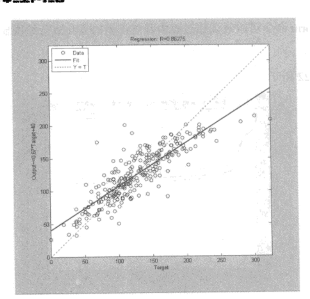


图 10-24 ldl 线性回归结果

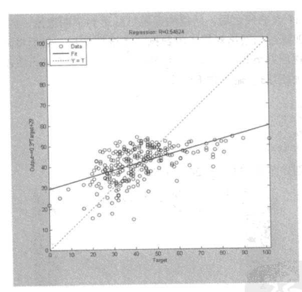


图 10-25 vldl 线性回归结果

可以看到，图 10-23、10-24 中的期望响应与仿真输出回归效果较好，相关系数都在 0.9 左右，而图 10-25 输出则与期望值吻合得不是很好，相关系数只有 0.54。通过修改网络结构（例如增加隐层神经元数目），或利用其他训练方法，可以改善网络运行的效果。

10.5.3 特征识别

利用计算机进行模式识别是一项很有用的技术，尤其是利用机器来识别图形符号的

特征，如银行支票的签字，这是非常省时省力的，同时避免了人工操作可能出现的很多问题。

【例 10-16】 利用 BP 网络进行特征识别实例。设计并训练一个 BP 网络，完成 26 个字母的图像识别。

解： ① 载入字母表。英文字母可以通过 5×7 像素的二值数字图像来表示。图 10-26 所示为字母 A 的对应图像。

将 5×7 个像素对应的值转化为一维向量，则每个字母对应的向量含有 35 个元素，从而组成了一个向量矩阵 alphabet。而期望输出向量具有 26 个元素，该字母在字母表中所处位置的元素为 1，其他位置的元素为 0。

例如，字母 A 在字母表中为第一个字母，因此其期望输出向量为[1,0,0,…,0]。下面载入字母表。输入命令：

```
clear
[alphabet,targets] = prprob;
[R,Q] = size(alphabet);
[S2,Q] = size(targets);
```

其中 prprob 文件定义了 26 个字母对应的图像向量以及目标向量。由于现实中的字符图像并不一定是理想的，因此网络必须具有一定的容错能力。含有噪声的字母 A 的图像如图 10-27 所示。

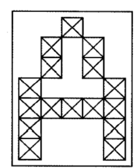


图 10-26 字母 A 的图像

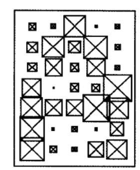


图 10-27 含有噪声的字母 A 图像

② 创建网络并初始化。网络接收 35 个布尔数值作为输入，也即输入向量含有 35 个元素，网络输出就是所在位置的具有 26 个元素（26 个字母）的输出向量。对于正确的网络，应该能够产生与输入对应的输出响应。

同时网络还应该具有一定的容错能力。当噪声均值为 0，标准差小于 0.2 时，网络应该仍然能够正确地识别输入字母向量。

利用 newff 函数设计一个两层的神经网络，其中第一层隐层中含有 10 个神经元，这个值是根据经验和问题复杂度的估计而定的。如果识别效果不理想的话，可以进一步增加神经元的数目。第一、二层神经元均采用 logsig 传递函数，对应的网络结构如图 10-28 所示。

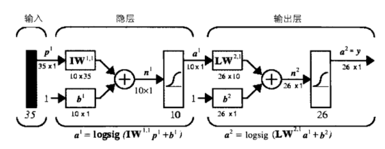


图 10-28　BP 网络结构

输入下列命令创建网络：

```
S1 = 10;
net = newff(minmax(alphabet),[S1 S2],{'logsig' 'logsig'},'traingdx');
net.LW{2,1} = net.LW{2,1}*0.01;
net.b{2} = net.b{2}*0.01;
```

③ 对网络进行训练。

设定网络训练函数以及训练样本向量，为了使网络具有容错能力，能够处理带有噪声的输入样本，需要利用理想样本和噪声样本对网络进行训练。因此训练过程这样进行：首先利用理想样本向量进行训练，直到达到期望的误差值。输入如下命令：

```
net.performFcn = 'sse';
net.trainParam.goal = 0.1;
net.trainParam.epochs = 5000;
net.trainParam.mc = 0.95;
P = alphabet;
T = targets;
[net,tr] = train(net,P,T);
```

训练过程中网络误差的变化情况如图 10-29 所示。

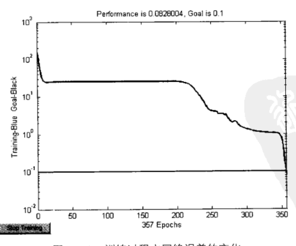


图 10-29　训练过程中网络误差的变化

④ 建立一个网络副本，对网络副本进行含噪声的样本训练，设定期望误差为 0.6，最大迭代次数为 300。输入命令：

```
netn = net;
netn.trainParam.goal = 0.6;
netn.trainParam.epochs = 300;
```

定义网络输入样本为理想样本与含有随机噪声的样本合成的向量，然后开始对网络训练，重复训练 10 次，这样做的目的是使得训练产生的网络既能够识别理想输入样本，也能够识别含噪声的输入样本。输入命令：

```
T = [targets targets targets targets];
for pass = 1:10;
  fprintf('Pass = %.0f\n',pass);
  P = [alphabet, alphabet, ...
      (alphabet + randn(R,Q)*0.1), ...
      (alphabet + randn(R,Q)*0.2)];
  [netn,tr] = train(netn,P,T);
end
```

然后再次进行无噪声样本的训练，这样做的目的是使得网络对理想信号的识别更为迅速。输入命令：

```
netn.trainParam.goal = 0.1;
netn.trainParam.epochs = 500;
P = alphabet;
T = targets;
[netn,tr] = train(netn,P,T);
```

⑤ 最后利用输入样本向量对网络进行仿真，并测试输出。输入命令如下：

```
noise_range = 0:.05:.5;     %设定输入噪声大小
max_test = 100;
network1 = [];
network2 = [];
T = targets;

for noiselevel = noise_range   %设置主循环
  fprintf('Testing networks with noise level of %.2f.\n',noiselevel);
  errors1 = 0;
  errors2 = 0;

  for i=1:max_test
    P = alphabet + randn(35,26)*noiselevel;          %添加噪声

    A = sim(net,P);                     %对无噪样本训练网络进行仿真
    AA = compet(A);                     %竞争输出，获得二值字母图像输出
    errors1 = errors1 + sum(sum(abs(AA-T)))/2;       %计算误差

    An = sim(netn,P);                   %对含噪样本训练网络进行仿真
    AAn = compet(An);                   %竞争输出，获得二值字母图像输出
```

```
    errors2 = errors2 + sum(sum(abs(AAn-T)))/ 2;    %计算误差
  end
  network1 = [network1 errors1/26/100];      %记录无噪样本训练网络误差数据
  network2 = [network2 errors2/26/100];      %记录含噪样本训练网络误差数据
end

plot(noise_range,network1*100,'--',noise_range,network2*100,'*');
title('误差百分数');
xlabel('噪声水平');
ylabel('Network 1 - -   Network 2 **');
```

最终绘出的曲线如图 10-30 所示。

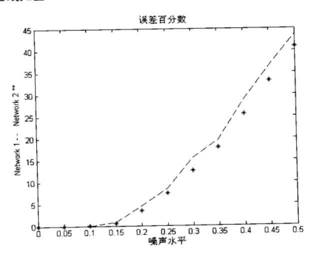


图 10-30　字符识别误差与输入噪声水平的关系

图 10-30 中，虚线表示的是用理想样本训练的网络，星号“*”表示的是用噪声样本训练的网络。可以看到，在输入噪声水平低于 0.1 的情况下，两个网络都可以很好地完成对字符的识别，在噪声水平较高的情况下，两个网络都不能够很好进行字符识别，但是经过噪声样本训练后的网络，其容错能力比未经过噪声样本训练的网络要强。如果希望网络具有更高的容错性，可以在训练中增加噪声的比重。

需要说明的是，由于版本差异，此例程在 MATLAB 7.0 版本中是运行无误的，但在 MATLAB R2009b 版本中执行结果，可能会有所不同。

10.6 小结

本章首先介绍 BP 网络的结构和学习规则，然后结合实例介绍如何利用 MATLAB 工具箱函数创建、训练 BP 型神经网络。

第 11 章　径向基神经网络

1985 年，Powell 提出了多变量插值的径向基函数方法(Radial-based Function Method)。1988 年，Broomhead 和 Lowe 在此基础上提出了径向基神经网络，也就是 RBF 神经网络结构。径向基神经网络采用高斯函数等径向基函数作为神经元传递函数，能够实现非线性关系的映射。径向基神经网络在函数逼近、时间序列分析、非线性控制、模式分类、图像处理等方面有很广泛的应用。

结构上，径向基神经网络仍然属于多层前向型神经网络，通常由隐层、输出层两个神经元层组成。径向基神经网络相对于 BP 神经网络而言往往需要更多的神经元，但是它的训练速度更快，在输入向量样本数目较多的情况下，径向基网络的效果是很好的。

11.1 基本径向基神经网络

径向基神经元通常采用径向基传递函数，网络结构上通常包含两层神经元，第一层为隐含的径向基层，第二层为输出线性层。本节我们将对其神经元和网络结构进行介绍。

11.1.1 径向基网络神经元模型

图 11-1 给出了一个具有 R 个输入的径向基神经元模型结构图。图中每一个输入被赋予一定的权值，与偏差求和后作为神经元传递函数的自变量，神经元的输出是传递函数的输出。径向基神经元具有多种不同的传递函数，最常用的传递函数是高斯函数。

与其他类型的神经元有所不同的是，径向基神经元传递函数的输入是权值向量和输入向量之间的向量距离与偏差 b 的乘积。图中‖ dist ‖表示的是权值与输入向量的点积。

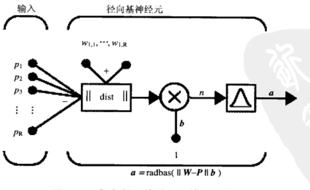


图 11-1　径向基网络神经元模型示意图

高斯型径向基神经元传递函数的表达式为：

$$\mathrm{radbas}(n)=\mathrm{e}^{-n^2}$$

图 11-2 是此传递函数的图形。

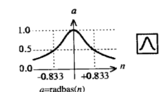


图 11-2　高斯型径向基神经元传递函数

当径向基传递函数的输入为 0 时，此径向基函数取得其最大值 1。随着权值向量 $\boldsymbol{w}$ 与输入向量 $\boldsymbol{p}$ 之间的距离的减小，输出逐渐增大。因此，对于一个径向基神经元，当输入 $\boldsymbol{p}$ 与权值 $\boldsymbol{w}$ 完全相等时，此神经元的输出恰好为 1。

偏差 b 的作用是调节径向基神经元的灵敏度。例如，如果一个神经元的偏差等于 0.2，那么对于输入向量 $\boldsymbol{p}$ 与输出向量 $\boldsymbol{w}$ 距离相差 8.326(0.8326/b)的情况，其输出值为 0.5。

11.1.2　径向基神经网络结构

径向基网络包括两层神经元，第一层是由 S^1 个径向基神经元组成的隐层，第二层是由 S^2 个线性神经元组成的输出层。其网络结构如图 11-3 所示。

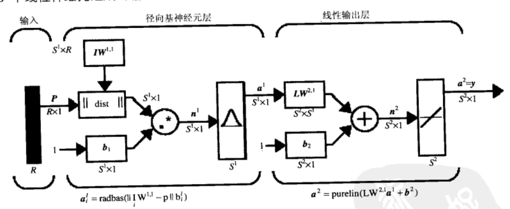


图 11-3　径向基神经网络结构

图 11-3 中，a_i^1 是向量 $\boldsymbol{a}^1$ 的第 i 个元素，${}_i\boldsymbol{IW}^{1,1}$ 是权值矩阵 $\boldsymbol{IW}^{1,1}$ 的第 i 个行向量。R 为输入向量元素的数目。‖ dist ‖标号的输入为输入向量 $\boldsymbol{p}$ 和输入权值矩阵 $\boldsymbol{IW}^{1,1}$，产生的输出向量包含 S_1 个元素，其中各个元素是输入向量 $\boldsymbol{p}$ 与矩阵 $\boldsymbol{IW}^{1,1}$ 各行之间的向量距离。

偏差向量 $\boldsymbol{b}^1$ 与‖ dist ‖相乘的结果可以由 MATLAB 运算符“.*”得到，该运算符表示的

是向量元素对元素的乘积。

前向网络的第一个神经元层的输出可以表示如下：

a{1} = radbas(netprod(dist(net.IW{1,1},p), net.b{1}))

幸运的是，实际上并不需要输入如此复杂的公式以获得这个计算结果。创建网络时所有的操作细节都由 MATLAB 工具箱函数 newrbe 或 newrb 直接完成，通过 sim 函数进行仿真即可得到输出结果。

我们可以通过跟踪一个输入向量 $\boldsymbol{p}$ 与输出向量 $\boldsymbol{a}^2$ 的关系来观察网络是怎样工作的。每一个径向基神经元层的神经元都会根据输入向量与神经元权值向量的距离产生一个输出。因此，如果径向基神经元权值矩阵与输入向量 $\boldsymbol{p}$ 相差非常大，就会产生接近于 0 的输出，对于线性输出层神经元而言，这些接近于 0 的输入值是可以忽略的。而如果径向基神经元权值矩阵与输入向量比较接近，就会产生接近于 1 的输出，对于线性输出层而言，这些接近于 1 的输入则是比较重要的。

实际上，如果一个径向基神经元的输出为 1，而其他所有的径向基神经元输出均为 0、或非常接近于 0 的话，线性神经元层的输出就完全是这个被激活为 1 的神经元的贡献。这是一种极端情况。典型的情况是，一部分神经元贡献比较大，另一部分则贡献较小。

现在来看看第一个神经元层具体是怎样工作的。每一个神经元的加权输入正是输入向量和它的权值向量的距离，该距离利用||dist||计算得到，也就是说每个神经元的网络输入是输入向量与权值向量的对应元素乘积之和，然后加上偏差的值。

在 MATLAB 中，这可以由函数 netprod 计算得到。每个神经元||dist||的输出都经过 radbas 函数处理，形成最终的输出。如果一个神经元的权值向量与输入向量相等的话，其网络输入就为 0，输出就为 1；而如果网络输入为 1 的话，其输出就为 0。如果权值向量和输入向量之间的距离等于散布常数 spread 的值，则传递函数输入为 0.8326，其输出就为 0.5。

11.2 概率神经网络

概率神经网络是径向基神经网络的一种，与基本径向基神经网络的不同在于它的第二层采用了竞争函数作为输出传递函数。概率神经网络通常用于模式分类问题。

在实际应用中，当概率神经网络获得一个输入时，第一层神经元计算输入向量与输入样本向量的距离，并产生一个向量，此向量的各个元素表征输入向量与输入样本向量的接近程度。

第二层神经元将与输入向量的各种类别综合起来，产生一个表征概率的输出向量。最后，一个竞争型（compete）的传递函数在输出端选择具有最大概率的输入向量类别，产生输出 1，而其他类别的输入向量则产生输出 0。其网络结构如图 11-4 所示。

R 为输入向量元素的数目；Q 为输入目标样本的数目，也是第一层神经元的数目；K 是输入向量类别的数目，也是第二层神经元的数目。

假设有 Q 个输入向量与目标输出向量对，每个目标向量有 K 个元素。目标向量中仅

一个元素为 1，其余的全为 0。这样，每一个输入向量与此 K 类中的一类相对应。

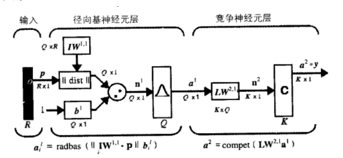


图 11-4　概率神经网络结构

图 11-4 中，a_i^1是向量$\boldsymbol{a}^1$的第 i 个元素，$_i\boldsymbol{IW}^{1,1}$是权值矩阵$\boldsymbol{IW}^{1,1}$的第 i 个行向量。将概率神经网络的第一层径向基神经元的输入权值矩阵$\boldsymbol{IW}^{1,1}$设为 Q 个训练样本对的转置$\boldsymbol{P}'$。当网络获得输入时，由|| dist ||计算得到一个向量，此向量的元素表征输入向量与训练样本的接近程度。然后该向量元素与偏差向量 $\boldsymbol{b}$ 的元素逐个相乘，结果传递给 radbas 函数。经过 radbas 计算之后，视输入向量与哪一个样本向量最接近，则$\boldsymbol{a}^1$中对应的元素就接近于 1。如果输入向量与一系列训练样本向量都接近的话，则对应的几个元素都接近于 1。

概率神经网络的第二层权值矩阵 $\boldsymbol{LW}^{1,2}$ 设为期望目标响应 $\boldsymbol{T}$。权值矩阵的每一个行向量只有一个元素为 1，对应着一类输入，其余元素均为 0，然后计算矩阵乘积 $\boldsymbol{Ta}^1$。最后，第二层神经元采用竞争传递函数计算 $\boldsymbol{n}^2$，对其中最大的元素输出为 1，其余元素取为 0。从而，概率神经网络就完成了对输入向量的划分。

11.3 广义回归神经网络

广义回归神经网络（GRNN）是由一个径向基层与一个特殊线性层构成的神经网络，通常被用来做函数逼近。GRNN 的网络结构如图 11-5 所示，它在网络结构上与标准的径向基神经网络有一些细微的差别。

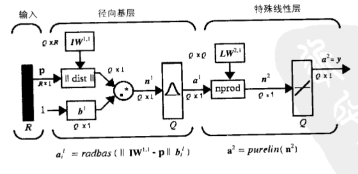


图 11-5　广义回归神经网络结构

图 11-5 中，a_i^1 是向量 $\boldsymbol{a}^1$ 的第 i 个元素；$\underset{i}{\boldsymbol{IW}}^{1,1}$ 是权值矩阵 $\boldsymbol{IW}^{1,1}$ 的第 i 个行向量；R 为输入向量元素的数目；Q 为输入输出样本的数目，也是第一、二层神经元的数目。

此外，图中 "nprod" 方框的输出为的 S^2 维向量 $\boldsymbol{n}^2$，其中的元素是权值矩阵 $\boldsymbol{LW}^{2,1}$ 各行与输入向量 $\boldsymbol{a}^1$ 的点积，结果根据 $\boldsymbol{a}^1$ 的元素和进行归一化。假设：

```
LW{2,1}= [1 -2;3 4;5 6];
a{1} = [0.7;0.3];
```

则有：

```
aout =.normprod(LW{2,1},a{1})
aout = 0.1000
       3.3000
       5.3000
```

广义回归神经网络的第一层与通常的径向基神经网络相同。其神经元数目与输入目标向量对 $\boldsymbol{P}$ 的个数相等。第一层的权值设为 $\boldsymbol{P}'$，偏差 $\boldsymbol{b}^1$ 设为 0.8326/spread 的列向量。由用户选择 spread 矩阵，应该使得输入向量与神经元权值向量的距离为 0.5。

此外，广义回归神经网络第一层神经元的功能也与通常的径向基神经元相同。每个神经元的加权输出结果就是由‖dist‖计算得到的输入向量与权值向量的距离。每一个神经元传递函数的输入是偏差与加权后输入向量的乘积，输出则是 radbas 函数计算得到的结果。

如果神经元的权值向量与输入向量的转置相等，则其加权后的结果为 0，也即传递函数输入为 0，输出结果为 1。如果权值向量和输入向量之间的距离等于散布常数 spread 的值，则传递函数输入为 0.8326，其输出就为 0.5。

广义回归神经网络的第二层包含的神经元数目与输入向量/目标向量的数目一样，但是权值矩阵 $\boldsymbol{LW}^{2,1}$ 的值被设定为 $\boldsymbol{T}$。

假定输入向量为 $\boldsymbol{p}$ 接近于样本输入向量的 $\boldsymbol{p}_i$，这个输入 $\boldsymbol{p}$ 产生的第一层输出 $\boldsymbol{a}^i$ 接近于 1，这使得第二层的输出接近于目标向量中的 $\boldsymbol{t}_i$ 的值，更大的散布常数会使得输入向量有更大的变动范围，在此范围内第一层神经元会产生较大的输出。如果散布常数较小，则径向基函数的曲线就会更陡，以致与权值向量最接近的输入向量所产生的输出会远大于其他的输入，此时网络就只对最接近权值向量的输入有响应。

随着散布常数逐渐增大，径向基函数的斜率会变得越来越平坦，这时将会有更多的神经元对输入向量有响应，此时网络的作用就相当于与输入向量接近的样本向量的加权平均。

11.4 径向基网络的 MATLAB 实现

本节我们通过实例对径向基神经网络、概率神经网络以及广义回归神经网络相关的 MATLAB 工具箱函数进行介绍。

11.4.1 径向基神经网络的精确创建

在 MATLAB 中，通过调用函数 newrbe 可以精确地生成一个径向基神经网络，此函数对于给定的训练向量能够生成一个零误差的网络对象。其调用格式如下：

net = newrbe(***p***, ***t***, spread)

其中，***p***、***t*** 分别为输入与输出样本向量，spread 为径向基神经元层的散布常数。该函数返回的网络对象对于给定的输入 ***p*** 具有精确的输出响应 ***t***。

newrbe 函数生成的神经元输入与 ***p*** 中向量个数相等，第一个层神经元权值设为 ***p***'。例如：有 Q 个输入向量，就会有 Q 个神经元。因此，网络中有一个具有 radbas 传递函数的径向基神经元层，能够对不同的输入向量起到检测的作用。

第一层径向基神经元层的偏差均被设置为 0.8326/spread。也就是说，对于加权后的绝对值大小等于 spread，则径向基函数的输出为 0.5，这决定了网络会响应的输入向量范围的大小。

如果 spread 的值为 4，则径向基神经元只有对那些与权值向量距离小于 4 的输入向量才会产生大于或等于 0.5 的响应。此外，散布常数 spread 应该足够的大，以使得神经元能够对整个输入空间产生响应。

生成的网络第二层为线性神经元层，其权值 $\boldsymbol{IW}^{2,1}$ 以及偏差 $\boldsymbol{b}^2$ 对于前一层的输出 $\boldsymbol{a}^1$，应当满足下面的线性方程：

[W{2,1} b{2}] * [A{1}; ones] = T

其中，W{2,1}、b{2}、A{1}分别是权值 $\boldsymbol{IW}^{2,1}$、偏差 $\boldsymbol{b}^2$，以及第一层输出 $\boldsymbol{a}^1$ 的对应 MATLAB 代码，T 为目标响应，而 A{1}和 T 都是已知的，所以可以应用下面的公式来计算第二层权值和偏差，计算结果具有最小的均方误差：

Wb = T/[P; ones(1,Q)]

此处，Wb 矩阵包含权值与偏差，偏差位于矩阵的最后一列，其均方误差始终为 0，解释如下。

此定解问题具有 C 个约束，但是有 C +1 个待求解变量，包括 C 个径向基神经元的 C 个权值，以及一个偏差；而一个有 C 个约束、而变量个数大于 C 的线性问题有无限个解。这也就是 newrbe 函数能够以零误差实现对输入和输出样本向量分类的原因。

因此，利用 newrbe 函数能够生成一个对于给定的训练样本向量具有零误差的网络。唯一的条件是需要给出一个足够大的散布常数 spread，以使得神经元响应的输入范围能够覆盖足够大的区域。这样对于任意时刻的任意输入，总有几个径向基神经元能够产生较大的响应输出。这样会使得网络传递函数曲线更加光滑，能够更好地综合反映不同于样本向量的新的输入向量。不过，spread 常数也不能够太大，以致各个神经元都具有完全重叠的输入向量响应区域，反而会造成精度的问题。

需要指出的是，newrbe 函数所生成的神经网络隐层中神经元的数目与样本输入向量的

数目一样多。因此，在样本输入向量数目非常多的场合，应用 newrbe 函数就不合适了。

11.4.2 更有效的径向基神经网络创建

采用 MATLAB 工具箱提供的函数 newrb 可以以迭代的方式生成一个径向基神经网络。在网络的创建过程中，神经元会不断地加入到网络之中，直到均方误差下降到期望的误差之下，或者网络达到最大神经元数目为止。newrb 函数的调用方式如下：

net = newrb(***p***, ***t***, goal, spread)

其中，***p***，***t*** 分别为输入与输出样本向量，spread 为径向基神经元层的散布常数，goal 为设定的网络误差目标。该函数返回的网络对象对于给定的输入 ***p*** 具有输出响应 ***t***，误差精度在 goal 以内。

newrb 函数与 newrbe 相似，不同之处在于，newrb 采用每次添加一个神经元的方式工作。每次迭代时，利用输入向量中对降低网络输出误差最有效的那一个来生成一个径向基神经元。

随后检查新生成网络的误差，如果足够小的话，newrb 函数就结束；否则，继续添加新的神经元。这个过程会持续到网络达到设定的误差目标，或者网络达到最大神经元数目为止。

同 newrbe 函数一样，应用 newrb 函数也要小心地选择散布常数，既要大到使得神经元产生响应的输入范围能够覆盖足够大的区域，同时也不能太大，而使各个神经元都具有重叠的输入向量响应区域。

为什么不用径向基神经网络取代标准的前向型网络呢？原因是，即使采用最有效的方法设计径向基神经网络，相比使用 tansig 或 logsig 传递函数的前向型神经网络，其隐层也需要更多的神经元。

这是因为 sigmoid 神经元能够对输入空间中的较大范围产生响应，然而 radbas 神经元只能够对输入空间中相对比较小的范围产生响应。结果是，输入空间越大（也即输入向量越多，或者说输入向量变动范围越大），网络所需要的 radbas 神经元就越多。

不过好的方面是，设计一个径向基神经网络通常所需要的时间要比设计一个 sigmoid 线性神经网络要短，并且有时所需的神经元数目也更少。

11.4.3 概率神经网络的创建

MATLAB 神经网络工具箱提供了函数 newpnn 用于创建一个概率神经网络，下面举例说明。

【例 11-1】 概率神经网络创建实例。给定 7 个输入和目标响应样本向量，应用 newpnn 函数生成一个与之对应的概率神经网络。

解： ① 首先定义输入与目标响应样本向量对。输入命令：

```
P = [0 0;1 1;0 3;1 4;3 1;4 1;4 3]',
```

```
Tc = [1 1 2 2 3 3 3],
```

输出结果为：

```
P = 0    1    0    1    3    4    4
    0    1    3    4    1    1    3
Tc =1    1    2    2    3    3    3
```

② 我们需要将序号转换为向量，这可以通过 ind2vec 函数来完成。输入命令：

```
T = ind2vec(Tc),
```

输出结果为：

```
T =
   (1,1)        1
   (1,2)        1
   (2,3)        1
   (2,4)        1
   (3,5)        1
   (3,6)        1
   (3,7)        1
```

③ 现在可以通过 newpnn 函数建立概率神经网络并对其进行仿真了，仿真输入设定为 ***P***，需要将向量格式的输出 ***Y*** 转换为标量格式 Yc。在命令行输入：

```
net = newpnn(P,T);
Y = sim(net,P);
Yc = vec2ind(Y)
```

输出结果为：

```
Yc =    1    1    2    2    3    3    3
```

④ 我们也可以尝试对样本向量以外的输入进行分类，例如对于下面的输入向量 P2，进行仿真。输入命令：

```
P2 = [1 4;0 1;5 2]',
Y = sim(net,P2);
Yc = vec2ind(Y)
```

得到的结果为：

```
P2 =1    0    5
    4    1    2
Yc =2    1    3
```

结果是正确的，因为仿真输入的向量确实分别与样本输入的 2，1，3 类比较接近。网络能够正确地对不同于输入样本的向量进行分类。

11.4.4 广义回归神经网络的创建

利用 MATLAB 工具箱函数 newgrnn 可以创建一个广义回归神经网络。下面举例说明。

【例 11-2】 广义回归神经网络创建实例。给定输入和目标响应样本向量，应用 newgrnn 函数生成一个与之对应的 GRNN 神经网络。

解：首先定义输入与目标响应样本向量。输入命令：

```
P = [4 5 6];
T = [1.5 3.6 6.7];
```

然后创建 GRNN 神经网络。输入命令：

```
net = newgrnn(P,T);
```

对网络进行仿真。输入命令，

```
P = 4.5;
v = sim(net,P),
```

输出结果为：

```
v =    3.0111
```

以上就完成了一个简单 GRNN 神经网络的创建和仿真。

11.5 径向基网络设计实例

径向基网络可以应用于函数逼近、模式分类等诸多方面。这一节我们将对其应用实例进行介绍。

11.5.1 径向基网络函数逼近

【例 11-3】 径向基网络函数逼近实例。应用 newrb 函数生成一个径向基神经网络，对给定的输入与输出数据点集关系进行函数拟合。

解：① 定义输入和输出向量，共 21 对，并作图。输入如下命令：

```
P = -1:.1:1;
T = [-.9602 -.5770 -.0729  .3771  .6405  .6600  .4609 ...
     .1336 -.2013 -.4344 -.5000 -.3930 -.1647  .0988 ...
     .3072  .3960  .3449  .1816 -.0312 -.2189 -.3201];
plot(P,T,'+');
title('训练样本（Training Vectors）');
xlabel('输入样本向量（Input Vector P）');
ylabel('目标向量（Target Vector T）');
```

绘出的曲线如图 11-6 所示。

② 对这 21 对输入、输出样本点进行拟合，可以通过径向基网络实现。径向基网络为两层结构，第一层为径向基神经元隐层，第二层为线性输出层。

创建径向基神经网络，并绘图显示其径向基传递函数。输入如下命令：

```
p = -3:.1:3;
a = radbas(p);
```

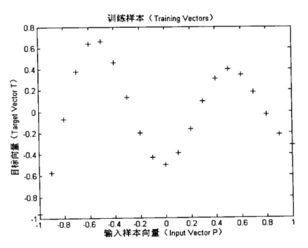


图 11-6　输入输出样本向量

```
plot(p,a)
title('径向基传递函数');
xlabel('输入 p');
ylabel('输出 a');
```

绘出的图形如图 11-7 所示。

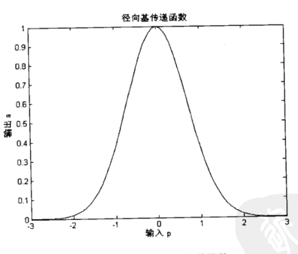


图 11-7　径向基神经元传递函数

③ 隐层中的各个神经元的权值和偏差决定了径向基函数曲线的宽度和位置，而输出层各个线性神经元对径向基神经元的输出进行加权求和。

如果每一层的权值和偏差值都正确，网络将在期望的精度内实现对函数的拟合。如图 11-8 所示，下面以三条曲线为径向基传递函数曲线，三条曲线相加得到对输入、输出样本的拟合曲线（虚线）。输入命令：

```
a2 = radbas(p-1.5);
a3 = radbas(p+2);
a4 = a + a2*1 + a3*0.5;
plot(p,a,'b-',p,a2,'b-',p,a3,'b-',p,a4,'m--')
title('径向基传递函数加权和');
xlabel('输入 p');
ylabel('输出 a');
```

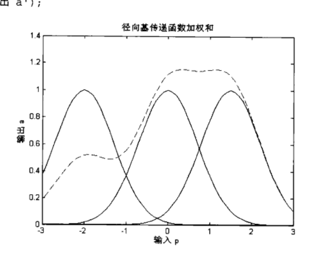


图 11-8 径向基传递函数加权和

在上面的步骤中，我们通过直接指定 3 个径向基传递函数的加权求和，对 ***P*** 和 ***T*** 进行了初步的逼近，可以看到，这样逼近的效果是粗糙的。

④ 下面应用函数 **newrb**，快速地创建一个径向基神经网络，并且通过预先设定期望的均方误差以及散布常数，可以以任意给定的精度对函数进行逼近。输入命令：

```
eg = 0.02; % 期望均方误差
sc = 1;    % 散布常数
net = newrb(P,T,eg,sc);
```

输出结果为：

```
NEWRB, neurons = 0, MSE = 0.176192
```

⑤ 为了查看网络的性能，可以重新仿真，并对训练结果重新绘图显示。输入命令：

```
plot(P,T,'+');
xlabel('Input');
X = -1:.01:1;
Y = sim(net,X);
hold on;
plot(X,Y);
hold off;
legend({'目标向量','网络输出'})
```

绘图结果如图 11-9 所示。

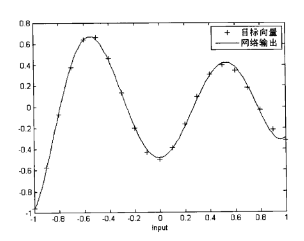


图 11-9　网络输出与目标向量比较

图 11-9 中，加号“+”代表的点为期望的目标输出，而曲线为径向基拟合曲线。可以看到，产生的径向基神经网络对期望的输入、输出样本对拟合得相当好。

11.5.2　散布常数的影响之欠交叠情形

在前面的章节中曾经提过，散布常数的选择对于径向基神经网络的创建是有很大影响的，如果散布常数选择不当，会造成网络中神经元响应区域不能覆盖整个输入范围，或者交叠区域过大导致重复响应，因而造成网络功能的不适性或者欠适性，这一节我们举例说明因为由于散布常数选取不当，导致网络神经元响应欠交叠对网络性能的情形。

【例 11-4】　欠交叠情形的散布常数影响实例。对【11-3】中给定的输入、输出数据点集，选择很小的散布常数，创建径向基神经网络进行函数拟合。

解： ① 定义输入和输出向量，共 21 对数据，并作图。输入如下命令：

```
P = -1:.1:1;
T = [-.9602 -.5770 -.0729  .3771  .6405  .6600  .4609 ...
     .1336 -.2013 -.4344 -.5000 -.3930 -.1647  .0988 ...
     .3072  .3960  .3449  .1816 -.0312 -.2189 -.3201];
plot(P,T,'+');
title('训练样本（Training Vectors）');
xlabel('输入样本向量（Input Vector P）');
ylabel('目标向量（Target Vector T）');
```

绘出的曲线如图 11-10 所示。

② 利用 newrb 函数快速创建径向基网络，对 ***P***、***T*** 函数关系进行逼近。此处函数参数中预先设定的期望均方误差以及散布常数都选择非常小的数。输入如下命令：

```
eg = 0.02; % 期望均方误差
sc = .01;  % 散布常数
```

```
net = newrb(P,T,eg,sc);
```

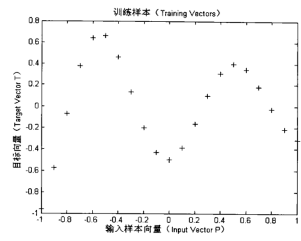


图 11-10 输入与输出样本向量

得到输出为：

```
NEWRB, neurons = 0, MSE = 0.176192
```

为查看网络的性能，用 sim 函数对其进行仿真，并对训练结果重新绘图显示。输入命令：

```
X=-1:.01:1;
Y=sim(net,X);
hold on;
plot(X,Y);
hold off;
```

绘出的图形如图 11-11 所示。

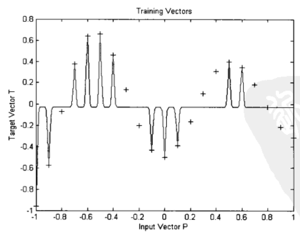


图 11-11 散布常数过小时，径向基神经网络函数逼近的结果

图 11-11 中，加号 "+" 代表的点为期望的目标输出，曲线为径向基拟合曲线。可以看到，由于散布常数选择得太小，产生的径向基神经网络对期望的输入、输出样本的拟合曲线中出现了很多不应有的尖峰，这种拟合的结果被称为过适性。如果选择一个大小合适的散布常数，则不会出现这样的结果。

11.5.3 散布常数的影响之过交叠情形

与上例相反，如果散布常数过大的话，会导致网络中神经元响应区域过交叠，造成网络的欠适性。这一节我们举例说明因为散布常数选取过大导致网络神经元响应过于交叠对网络性能造成的影响。

【例 11-5】 过交叠情形的散布常数影响实例。对【例 11-3】中给定的输入、输出数据点集，选择很大的散布常数，创建径向基神经网络进行函数拟合。

解： ① 定义输入、输出向量，共 21 对数据，并作图。输入如下命令：

```
P = -1:.1:1;
T = [-.9602 -.5770 -.0729  .3771  .6405  .6600  .4609 ...
     .1336 -.2013 -.4344 -.5000 -.3930 -.1647  .0988 ...
     .3072  .3960  .3449  .1816 -.0312 -.2189 -.3201];
plot(P,T,'+');
title('训练样本（Training Vectors）');
xlabel('输入样本向量（Input Vector P）');
ylabel('目标向量（Target Vector T）');
```

绘出的曲线如图 11-12 所示。

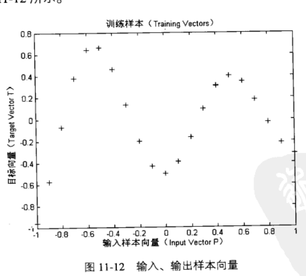


图 11-12 输入、输出样本向量

② 利用 newrb 函数快速创建径向基网络，对 ***P***、***T*** 函数关系进行逼近。此处函数参数中预先设定的期望均方误差以及散布常数都选择非常大的数。输入如下命令：

```
eg = 0.02; % 期望均方误差
sc =100;  % 散布常数
net = newrb(P,T,eg,sc);
```

得到输出为：

```
NEWRB, neurons = 0, MSE = 0.176192
```

为查看网络的性能，用 sim 函数对其进行仿真，并重新绘图显示训练结果。输入命令：

```
X=-1:.01:1;
Y=sim(net,X);
hold on;
plot(X,Y);
hold off;
```

绘出的图形如图 11-13 所示。

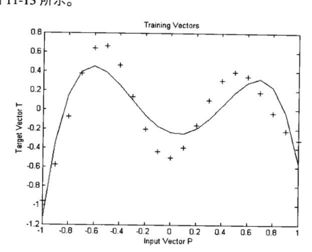


图 11-13 散布常数过大时径向基神经网络函数逼近的结果

图 11-13 中，加号 “+” 代表的点为期望的目标输出，曲线为径向基拟合曲线。可以看到，产生的径向基神经网络不能对期望的输入、输出样本对准确地拟合，拟合曲线的误差很大，这样的结果被称为不适性，这是散布常数选择过大造成的结果。

11.5.4 广义回归网络函数逼近

广义回归神经网络（GRNN）通常被用来做函数逼近，这一节对其举例进行说明。

【例 11-6】 广义回归网络函数逼近实例。对给定的输入、输出数据点集，创建 GRNN 径向基神经网络，并进行函数拟合。

解： ① 定义输入、输出向量，并作图。输入如下命令：

```
P = [1 2 3 4 5 6 7 8];
T = [0 1 2 3 2 1 2 1];
plot(P,T,'.','markersize',30)
```

```
axis([0 9 -1 4])
title('输入输出样本')
xlabel('P')
ylabel('T')
```

绘出的图形如图 11-14 所示。

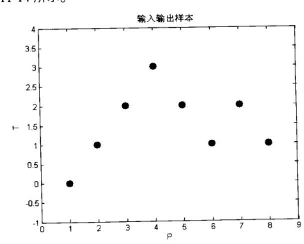


图 11-14　输入、输出样本向量

② 利用 newgrnn 函数创建广义回归径向基神经网络，对 ***P***，***T*** 函数关系进行逼近。选取的散布常数 spread 略小于输入向量之间的距离 1。散布常数比较小，对输入、输出样本点的拟合会比较好，但光滑度会差一些。输入如下命令：

```
spread = 0.7;
net = newgrnn(P,T,spread);
A = sim(net,P);
hold on
outputline = plot(P,A,'o','markersize',10,'color',[1 0 0]);
title('网络输出与样本比较')
xlabel('P')
ylabel('T and A')
```

绘出的图形如图 11-15 所示。

③ 在图 11-15 中，实心圆点代表输出样本点，空心圆点代表创建的 GRNN 仿真的输出。可以看到，网络对输入、输出样本函数进行了初步的逼近，虽然有一些误差，但基本上达到了拟合的效果。我们可以增加一些新的输入值，对网络进行仿真，并将其用加号“+”绘制到原图上。输入如下命令：

```
p = 3.5;
a = sim(net,p);
plot(p,a,'+','markersize',10,'color',[1 0 0]);
title('新输入值')
xlabel('P and p')
ylabel('T and a')
```

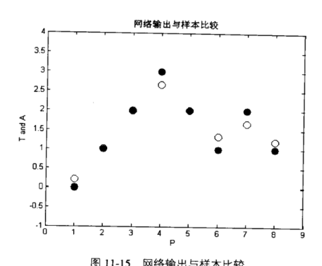


图 11-15 网络输出与样本比较

绘出的图形如图 11-16 所示。

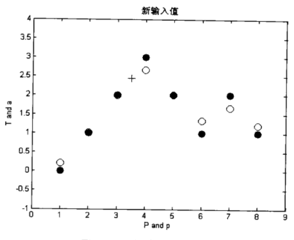


图 11-16 新输入样本的输出

图 11-6 中，新的样本点用加号 "+" 表示。下面增加更多的新输入样本点，将网络仿真响应用曲线连接起来。输入如下命令：

```
P2 = 0:.1:9;
A2 = sim(net,P2);
plot(P2,A2,'linewidth',4,'color',[1 0 0])
title('网络拟合的函数曲线')
xlabel('P and P2')
ylabel('T and A2')
```

绘出的曲线如图 11-17 所示。

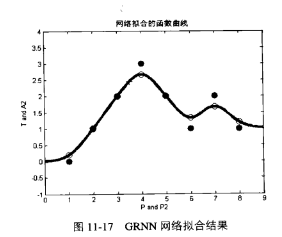


图 11-17　GRNN 网络拟合结果

11.5.5　概率神经网络模式分类

概率神经网络（PNN）通常被用来对输入样本进行，本节对其举例进行说明。

【例 11-7】　概率神经网络模式分类实例。对给定的输入、输出样本点集，创建 PNN 进行分类。

解： ① 定义输入、输出样本向量，并作图。注意此处输入样本向量为二维的，输出向量表征输入向量的所属类别。输入如下命令：

```
P = [1 2; 2 2; 1 1]';
Tc = [1 2 3];
plot(P(1,:),P(2,:),'.','markersize',30)
for i=1:3, text(P(1,i)+0.1,P(2,i),sprintf('class %g',Tc(i))), end
axis([0 3 0 3])
title('3 个样本向量')
xlabel('P(1,:)')
ylabel('P(2,:)')
```

绘出图形如图 11-18 所示，

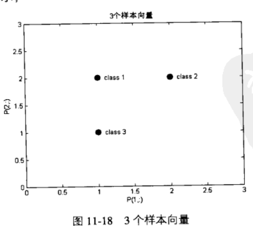


图 11-18　3 个样本向量

② 要应用函数 newpnn 创建概率神经网络，我们需要将标号转换为向量，这可以通过 ind2vec 函数来完成，同时设定网络散布常数等于 1，输入命令如下：

```
T = ind2vec(Tc);
spread = 1;
net = newpnn(P,T,spread);
```

下面对利用输入、输出样本对网络的输出进行仿真测试，此处需要将输出结果向量重新转换成为标号。输入如下命令：

```
A = sim(net,P);
Ac = vec2ind(A);
plot(P(1,:),P(2,:),'o','markersize',10)
axis([0 3 0 3])
for i=1:3,text(P(1,i)+0.1,P(2,i),sprintf('class %g',Ac(i))),end
title('网络仿真测试结果')
xlabel('P(1,:)')
ylabel('P(2,:)')
```

绘出图形如图 11-19 所示。

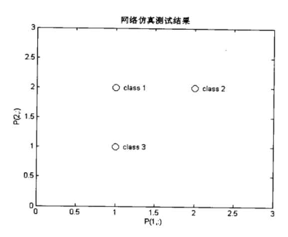


图 11-19　网络仿真测试结果

可以看到，仿真输出与期望输出是完全一致的。

③ 下面创建一个新的输入向量，应用创建的网络对其进行分类。输入命令：

```
p = [2; 1.5];
a = sim(net,p);
ac = vec2ind(a);
hold on
plot(p(1),p(2),'.','markersize',30,'color',[1 0 0])
text(p(1)+0.1,p(2),sprintf('class %g',ac))
hold off
title('新输入样本的分类仿真结果')
xlabel('P(1,:) and p(1)')
```

```
ylabel('P(2,:) and p(2)')
```

输出图形如图 11-20 所示。

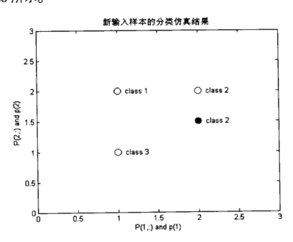


图 11-20 新输入样本的分类仿真结果

可以看到，新输入的向量被划分到了 class2。

④ 下面通过立体图的方式，显示创建的概率神经网络三个类别的分界区域。输入命令如下：

```
p1 = 0:.05:3;
p2 = p1;
[P1,P2] = meshgrid(p1,p2);          %将向量转化为坐标网格
pp = [P1(:) P2(:)]';
aa = sim(net,pp);                   %仿真
aa = full(aa);                      %将输出稀疏矩阵结构转化为满存储结构
m = mesh(P1,P2,reshape(aa(1,:),length(p1),length(p2)));    %绘制第一类
set(m,'facecolor',[0 0.5 1],'linestyle','none');
hold on                             %保持图形
m = mesh(P1,P2,reshape(aa(2,:),length(p1),length(p2)));    %绘制第二类
set(m,'facecolor',[0 1.0 0.5],'linestyle','none');
m = mesh(P1,P2,reshape(aa(3,:),length(p1),length(p2)));    %绘制第三类
set(m,'facecolor',[0.5 0 1],'linestyle','none');
plot3(P(1,:),P(2,:),[1 1 1]+0.1,'.','markersize',30)
plot3(p(1),p(2),1.1,'.','markersize',30,'color',[1 0 0])
hold off                            %解除保持
view(2)                             %设置由顶向下二维视角显示
title('3 类划分区域')
xlabel('P(1,:) and p(1)')
ylabel('P(2,:) and p(2)')
```

绘出的图形如图 11-21 所示。

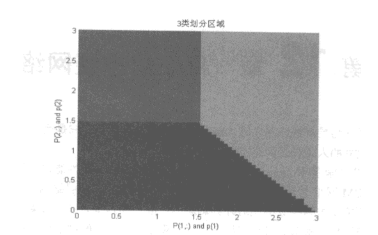


图 11-21 概率神经网络的模式分类情况

图 11-21 清楚地显示了应用此概率神经网络进行模式划分的决策线。

11.6 小结

本章首先介绍径向基网络神经元和网络结构，然后介绍两种变形的径向基网络，即广义回归神经网络和概率神经网络，最后通过实例介绍径向基网络的应用。



第 12 章　自组织神经网络

自组织神经网络是神经网络领域内最激动人心的方向之一，它是受生物神经系统结构和现象启发而建立的人工神经网络结构。

1981 年，芬兰学者 Kohonen 提出了一个较为完整的、分类性能较好的自组织特征映射神经网络（SQM 网络）。SQM 网络一般具有输入层和竞争层两层结构，其中竞争层为其核心层。网络模拟了大脑神经细胞对外界刺激的反应，通过输入样本反复的无监督学习，将输入模式的统计特征融解到各个连接权值上，实现特定区域的神经元对特定模式的输入产生响应的功能，同时具有很强的抗干扰能力。

自组织神经网络还有其他的种类，例如学习向量量化网络（LVQ 网络）等。LVQ 网络是一种利用有监督学习对竞争性网络进行训练的网络类型。在无监督学习的训练方式下，竞争层对输入样本进行模式分类只依赖输入向量之间的距离。如果两个输入向量的距离接近的话，就会被划分为同一类，而利用 LVQ 网络，则可以由用户定义标准，对输入向量进行分类。

自组织神经网络包括自组织竞争网络、自组织特征映射网络、学习矢量量化等网络结构形式，下面我们将对这几种形式的网络结构和学习算法进行介绍。

12.1 自组织竞争网络

12.1.1 自组织竞争网络结构模型

图 12-1 给出了一个具有 R 个输入的自组织神经网络结构图。自组织竞争网络通常包含两层神经元，第一层为输入层，第二层为竞争层，其中竞争层是其核心层。竞争层中‖dist‖的输入为输入向量 $\boldsymbol{p}$ 与输入层权值矩阵 $\boldsymbol{IW}^{1,1}$，输出结果为一个含有 S_1 个元素的向量，其中各元素为输入向量与权值矩阵的各个行向量 ${}_i\boldsymbol{IW}^{1,1}$ 之间的距离。

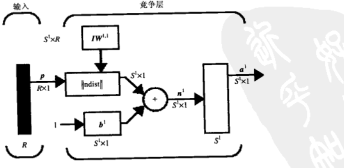


图 12-1　自组织网络结构示意图

通过计算输入向量 p 与权值向量 ${}_{i}IW^{1,1}$ 的距离的相反数，然后与偏差 b 求和得到结果 n^1，以此作为竞争层网络的输入。如果所有的偏差均为 0，则竞争层神经元的最大网络输入为 0，此时，网络输入 p 与权值向量恰好相等。

竞争传递函数接收上一层中各神经元的输出，当神经元竞争胜出时，输出为 1，其他所有竞争失败的神经元的输出均为 0。如果所有的偏差均为 0，则权值向量与输入向量的距离最小的神经元具有最小的负值网络输入，从而能够赢得竞争，输出为 1。

12.1.2 自组织竞争神经网络的学习算法

自组织竞争神经网络通常采用由内星学习规则发展来的 Kohonen 学习规则，下面对其进行介绍。

格劳斯贝格（S.Grossberg）为解释人类和动物的学习现象，提出了两种类型的神经元模型：内星（Instar）和外星（Outstar）。内星神经元通常被训练来识别一个向量，而外星神经元通常被训练来生成一个向量。

1．内星学习规则

内星神经元模型结构如图 12-2 所示。

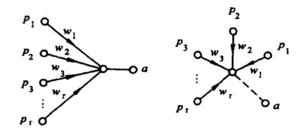


图 12-2 内星神经元模型结构

图 12-2 中的神经元具有 r 个输入，其各个输入值 p_i 组成输入向量 $\boldsymbol{P}$，通过权值向量 $\boldsymbol{W}$ 连接，形成网络输入。其网络结构呈星形，信号沿箭头方向流向网络的内部。

实现内星输入与输出转换的传递函数为 harlim 硬限值函数。通过内星及其学习规则可以训练，使得某一神经元节点只对特定的输入向量产生响应，这是借助于调节网络权值向量 $\boldsymbol{w}$ 近似于输入向量 $\boldsymbol{P}$ 来实现的。

图 12-2 中所示的单内星结构对权值进行修正的学习规则为：

$$\Delta w_{1j} = lr \cdot (p_j - w_{1j}) \cdot a$$

其中，$j = 1, 2, ..., r$，设此单内星的权值向量为 $\boldsymbol{W}_1$，w_{1j} 为 $\boldsymbol{W}_1$ 的第 j 个元素，p_j 为输入向量的第 j 元素，lr 为网络学习速率。

此规则称为格劳斯贝格内星学习规则。可以看到，在此学习规则下，内星神经元连接权值的变化 Δw_{1j} 是与输出成正比的。内星的传递函数即硬限值函数输出为 1 或 0，分别对应高值和低值，如果输出 a 被某一外部方式保持为高值时，通过反复的学习，就可以使变化量 Δw_{1j} 逐渐

减少，内星权值逐渐接近输入 p_j，最终达到 $w_{1j}=p_j$，从而使得内星权值向量 $\boldsymbol{W}_1$ 能够对输入向量 $\boldsymbol{P}$ 进行识别。然而，如果内星输出 a 保持为低值，则网络权值的学习过程可能无法进行。

考虑不同的输入向量 $\boldsymbol{P}^1$ 和 $\boldsymbol{P}^2$ 分别出现在同一个内星时的情况，此内星权值向量记为 $\boldsymbol{W}$。首先，为了训练需要，先要将输入向量进行归一化处理，对每一个输入向量除以所有向量的平方和，由此得到的新输入向量的模值为 1。

当第一个输入向量 $\boldsymbol{P}^1$ 输入给内星后，网络经过训练，权值向量达到了 $\boldsymbol{W}=(\boldsymbol{P}^1)^T$ 的状态。而后输入另一个向量 $\boldsymbol{P}^2$。此时内星的加权输入之和可由下式表示：

$$N=\boldsymbol{W}\cdot\boldsymbol{P}^2=(\boldsymbol{P}^1)^T\cdot\boldsymbol{P}^2=\left\|\boldsymbol{P}^1\right\|\cdot\left\|\boldsymbol{P}^2\right\|\cdot\cos\theta_{12}=\cos\theta_{12}$$

此加权输入就是新输入向量与原输入向量的点积，由于归一化后向量模值为 1，故加权输入就是两个输入向量之间的夹角余弦。

内星的加权输入和可分为 3 种情况。

- 情况 1：$\boldsymbol{P}^2=\boldsymbol{P}^1$，两者夹角 θ_{12}=0，此时内星加权输入和为 1。
- 情况 2：$\boldsymbol{P}^2$ 不等于 $\boldsymbol{P}^1$，随着 $\boldsymbol{P}^2$ 向着远离 $\boldsymbol{P}^1$ 的方向移动，内星加权输入和将逐渐减少，直到 $\boldsymbol{P}^2$ 与 $\boldsymbol{P}^1$ 垂直，即 θ_{12}=90°，此时内星加权输入和为 0。
- 情况 3：$\boldsymbol{P}^2=-\boldsymbol{P}^1$，即 θ_{12}=180° 时，内星加权输入和达到最小，取值为–1。

可见，对于一个经过训练的内星网络，当输入端再次出现训练样本向量的时候，内星将会获得值为 1 的加权输入；而如果输入为与训练样本不相同的向量时，产生的加权输入和总是小于 1 的。如果将内星加权输入和送给一个偏差略大于–1 的硬限制二值传递函数时，对于接近已学习的向量的输入，内星的输出为 1，其他的情况则输出为 0。

实际上，权值向量 $\boldsymbol{W}$ 与输入向量 $\boldsymbol{P}$ 的点积，就反映了输入向量与权值向量的相似度。当多个相似的输入向量输入内星时，最终的训练结果是使得网络的权值向量趋向于相似输入向量的平均值。

内星网络中的相似度是由偏差 b 来控制的。典型的相似度值为–0.95，这意味着输入向量与权值向量之间的夹角小于 18°48′。若 b=–0.9，则夹角扩大为 25°48′。

MATLAB 神经网络工具箱中内星学习规则对应的函数为 learnis。

2. 外星学习规则

外星神经元模型结构如图 12-3 所示。

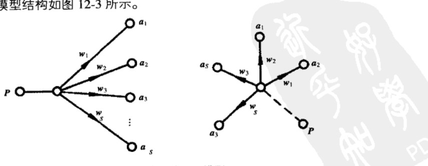


图 12-3　外星神经元模型

外星网络采用线性函数作为传递函数，它被用来学习和回忆一个向量，其网络输入 $\boldsymbol{P}$ 也可以是另一个神经元的输出。图中的外星网络具有 s 个输出，其所用的学习规则与内星规则也非常相似。其公式为：

$$\Delta \boldsymbol{w}_{i1} = lr \cdot (\boldsymbol{a}_i - \boldsymbol{w}_{i1}) \cdot \boldsymbol{p}_j$$

其中 $i = 1,2,...,s$ ， $j = 1,2,...,r$ ， r 为输入向量的维数；设单外星的权值向量为 $\boldsymbol{W}_1$ ， w_{i1} 为 $\boldsymbol{W}_1$ 的第 j 个元素，lr 为网络学习速率。此外，这里输入 p_j 是硬限值函数的结果，只取 0、1 两个值。

与内星不同的是，外星连接权值向量的变化 $\Delta \boldsymbol{W}_1$ 是与输入向量 $\boldsymbol{P}$ 成正比的，这意味着，当输入向量保持为高值时，外星的权值将趋于输出值。

当输入向量 p_j=0 保持为低值时，网络权值不进行学习与修正。

内星与外星之间的对称性是非常有用的。对一组输入和目标向量来训练一个内星层，若将其输入与目标向量对换，训练产生一个外星层，这两者的权值矩阵恰好是对方的转置。

MATLAB 神经网络工具箱中外星学习规则对应的函数为 learnos。

3．Kohonen 学习规则

Kohonen 学习规则是从内星规则发展而来的。在竞争型神经网络中，只对竞争中胜出的那一个神经元的权值进行调整。假定，第 i 个神经元在竞争中胜出，则其输入权值矩阵第 i 行就按照下面的公式进行调整：

$${}_i\boldsymbol{IW}^{1,1}(q) = {}_i\boldsymbol{IW}^{1,1}(q-1) + \alpha(\boldsymbol{p}(q) - {}_i\boldsymbol{IW}^{1,1}(q-1))$$

其中，q 为迭代次数，α 为调整系数，Kohonen 学习规则使得神经元权值矩阵可以学习一个输入向量，因此在模式识别中很有用处。在此学习规则之下，距离输入向量最近的神经元的权值向量将会更加接近输入向量。如果网络获得一个相似的输入向量，此神经元会更有更大的几率胜出，而如果网络获得一个不相似的输入向量的话，此神经元胜出的可能性将降低。

随着网络获得的输入向量越来越多，竞争层中每个距离这一组输入向量最近的神经元的权值将会向着这些输入向量的方向调整。最终，如果网络中有足够多的神经元的话，每一类相似的输入向量将会令一个神经元产生 1 的响应，而其他神经元产生 0 的响应。从而，此竞争型神经网络能够通过学习实现对输入向量的分类。

MATLAB 神经网络工具箱中对应 Kohonen 学习规则的函数为 learnk。

4．偏差学习规则

竞争型神经网络有一个局限性：网络中一部分神经元的权值向量的初始值可能与所有的输入向量都相差过大，因此在训练过程中永远都无法赢得竞争，因而不论训练持续时间多长，它们的权值向量都没有学习的机会。这些神经元被称为“死神经元”，是网络中有缺陷的部分。

为了防止这一点，可以对偏差进行调整，将正的偏差值加到负的距离上去，从而使得

那些在竞争中很少胜出的神经元能够获得更多的胜出机会。

在实际操作过程中，首先保留神经元输出的均值，它应该等于输出为 1 的神经元在所有神经元中所占的百分比，然后利用此平均值更新偏差值。MATLAB 神经网络工具箱中用来实现此功能的函数为 learncon。这个修正过程使得经常活动的神经元偏差越来越小，而那些不经常活动的神经元的偏差值则会变大。

随着不经常活动的神经元的偏差值的增大，它们能够响应的输入空间范围也逐渐增大。一旦输入空间范围增大了，那些不经常活动的神经元在竞争中胜出的机会也就增加了，从而逐渐达到和其他神经元一样活跃的目标。

此算法有两个好处。

（1）解决了“死神经元”的问题。如果一个神经元的权值向量与所有的输入向量都相距很远的话，将永远无法在竞争中胜出。然而在偏差学习规则下，其偏差值会逐渐增大，从而可能赢得竞争。当在竞争中胜出后，它的权值向量将会向某一组输入向量的方向移动，一旦它进入了某一输入向量组织后，就完全赢得了竞争，此时，其偏差值将下降为 0，从而解决“死神经元”的问题。

（2）可以使得每个神经元对具有相同百分比的输入向量进行大致的分类。如果输入空间中的一个区域中所包含的输入向量百分比较大，则其区域密度越大，就会吸引更多的神经元，从而对其进行更细致的分类。

竞争网络通常用于具有典型聚类特征的大量目标数据的识别，但是当遇到大量具有概率分布的输入向量时，竞争网络就显得力不从心了。这时候可以采用 Kohonen 自组织特征映射网络来处理，这是下一节将要介绍的内容。

12.2 自组织特征映射网络

自组织特征映射网络（Self-Organizing Feature Maps）又称做 SOFM 网络，是由芬兰赫尔辛基神经网络专家 Kohonen 在 1981 年提出的。这种网络模拟大脑神经系统自组织特征映射的功能。它是一种竞争型神经网络，采用无监督学习算法进行网络训练，此网络广泛地应用于样本分类、排序和样本检测等方面。本节对其网络模型与结构进行介绍。

12.2.1 自组织特征映射网络模型

脑科学的研究表明，人类大脑皮层中的细胞群存在着广泛的自组织现象。处于不同区域的神经元具有不同的功能，它们具有不同特征的输入信息模式，对不同感官输入模式的输入信号具有敏感性，从而形成大脑中各种不同的感知路径。并且这种神经元所具有的特性不是完全来自生物遗传，而是很大程度上依赖于后天的学习和训练。

在大脑皮层中，神经元的输入信号一部分来自感觉组织或其他区域的外部输入信号，另一部分则来自同一区域的反馈信号。神经元之间采用最邻近的神经元之间相互激励，次远的神经元相互抑制，而更远处的又具有一定的激励的方式进行信息交互，从而实现网络的自组织特性。

自组织映射模型具有这样的特点：输入节点与输出神经元的权值互联；在输出神经元之间进行竞争选择，输出神经元之间存在侧抑制。从功能上说，它能够将单个神经元的变化规则与一层神经元的群体变化规则联系在一起。

从网络结构上来说，其最大特点是神经元被放置在一维、二维或者更高维的网格节点上，图 12-4 是最普遍的自组织特征映射 SOFM 二维网格模型。

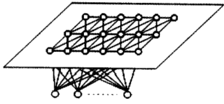


图 12-4 二维 SOFM 网格模型

SOFM 网络由输入层和竞争层组成，属于单层网络结构。输入层是一维的神经元，具有 N 个节点，竞争层的神经元处于二维平面网格节点上，构成一个二维节点矩阵，共有 M 个节点。输入层与竞争层的神经元之间都通过连接权值进行连接，竞争层邻近的节点之间也存在着局部的互连。SOFM 网络中具有两种类型的权值，一种是神经元对外部输入的连接权值，另一种是神经元之间的互连权值，它的大小控制着神经元之间相互作用的强弱。

在 SOFM 网络中，竞争层又是输出层，竞争层的神经元通常选择线性函数作为其传递函数，因此 SOFM 网络输出也就是各输入向量的加权和。

SOFM 网络通过引入网格形成了自组织映射的输出空间，并且在各个神经元之间建立了拓扑连接关系。神经元之间的联系是由它们在网格上的相互位置所决定的，这种联系模拟了人脑中的神经元之间的侧抑制功能，成为网络实现竞争的基础。

SOFM 网络采用的学习算法为无监督聚类法，它能将任意模式的输入在输出层映射成为一维或二维离散图形，并保持其拓扑结构不变。也就是说，在无监督的情况下，通过对输入模式的自组织学习，使得竞争层能够将输入的分类以二维图形的形式表示出来。

另一方面，通过反复的学习，SOFM 网络够使得连接权值空间分布密度与输入模式的概率分布趋于一致，因而可以通过连接权值的空间分布来反映输入模式的统计特征。

在 MATLAB 工具箱中，SOFM 网络中的神经元根据一些拓扑函数的设置而进行排列。比如函数 gridtop、hextop、randtop 分别可以将神经元按格点、六边形，以及随机拓扑进行排列。神经元之间的距离通过距离函数根据它们的位置计算得出。在 MATLAB 中，距离函数有 4 个，分别是：dist、boxdist、linkdist、mandist。其中 linkdist 是应用的最普遍的一个。

SOFM 网络采用与自组织竞争网络同样的方法来获得在竞争中胜出的神经元，此神经元用 i^*表示。但是，与自组织竞争网络不同的是，SOFM 网络采用的 Kohonen 学习规则不是仅仅更新胜出的单个神经元的权值，而是对其一定邻域 $N_{i^*}(d)$内的所有神经元的权值都进行更新。对权值进行更新的表达式如下：

$${}_i\boldsymbol{w}(q) = {}_i\boldsymbol{w}(q-1) + \alpha(\boldsymbol{p}(q) - {}_i\boldsymbol{w}(q-1))$$

或写做：

$${}_{i}\boldsymbol{w}(q)=(1-\alpha){}_{i}\boldsymbol{w}(q-1)+\alpha\boldsymbol{p}(q)$$

其中，邻域 $N_{i^*}(d)$ 中包含了所有与胜出神经元 i^* 的距离在一定范围之内的神经元。

$$N_i(d)=\{j,di_j\leqslant d\}$$

因此，当网络获得一个输入向量 $\boldsymbol{p}$ 时，竞争中胜出的神经元以及其邻域内的神经元的权值向量都会朝着输入向量 $\boldsymbol{p}$ 的方向移动。在经过多次反复训练之后，邻域内的神经元整体上也会变得比较相似。

另一种 SOFM 网络的训练规则，称为批处理算法。在这种训练规则下，网络一次获得整个输入样本集，然后根据训练规则决定对应各个输入向量哪一个神经元会胜出。胜出的神经元及其邻域内的所有神经元权值向量都朝着所有输入向量的平均位置进行移动。

为了对这个邻域的概念进行说明，让我们来看看图 12-5 的拓扑结构。左边的图中，在第 13 号神经元周围划出了一个二维的圆形邻域，其半径为 $d=1$。右图则在第 13 号神经元周围划出了一个半径 $d=2$ 的邻域。

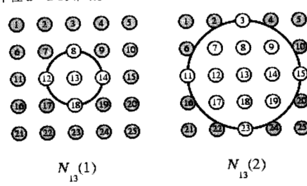


图 12-5　二维网格邻域示意图

其中，两个邻域分别可以写做：

$$N_{13}(1)=\{8,12,13,14,18\}$$

以及

$$N_{13}(2)=\{3,7,8,9,11,12,13,14,15,17,18,19,23\}$$

实际上，SOFM 网络中的神经元并不一定要排列成二维结构。同样可以采用一维排列方式，或者三维或更高维的排列形式。对于一个一维的 SOFM 网络，直径为 1 的邻域只包含两个最邻近的神经元（如果神经元位于端点的话，邻域只包含 1 个神经元）。

当然，也可以通过其他的方式来定义距离，例如，利用直角或者六角形的神经元排列。通常来说，网络的性能对于邻域的形状并不非常敏感。

下面对 SOFM 网络中的几种拓扑结构及其相关函数（gridtop、hextop、randtop）进行

介绍。

1．gridtop

格点拓扑结构函数 gridtop 给出的神经元排列在四方形的格点上，如图 12-5 所示，gridtop 函数的返回结果为表示神经元拓扑结构的矩阵。下面举例说明。

【例 12-1】 gridtop 函数应用实例。利用 gridtop 函数得到一个包含 6 个神经元的 2×3 的格点阵列。

解：直接调用 gritop 函数生成一个格点阵列拓扑结构的 SOFM 网络。输入命令：

```
pos = gridtop(2,3)
```

输出结果为：

```
pos =0    1    0    1    0    1
     0    0    1    1    2    2
```

这里，神经元 1 位于格点坐标(0,0)处，神经元 2 位于格点坐标(1,0)处，而神经元 3 位于格点坐标(0,1)处，其他神经元的位置依此类推，如图 12-6 所示。

注意，如果将 gridtop 函数的两个参数调换一下次序，得到的格点结构矩阵会有些许的不同。输入命令：

```
pos = gridtop(3,2)
```

输出结果为：

```
pos =0    1    2    0    1    2
     0    0    0    1    1    1
```

类似的，要得到一个包含 80 个神经元的 8×10 的格点阵列，并绘图显示，可以输入命令：

```
pos = gridtop(8,10);
plotsom(pos)
```

得到的输出结果如图 12-7 所示。

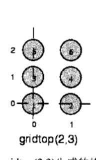


图 12-6 gridtop(2,3)生成的格点结构示意图

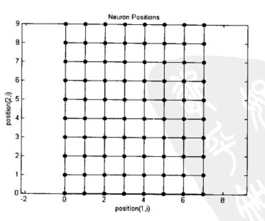


图 12-7 gridtop(8,10)生成的格点结构示意图

可以看到，所有神经元确实是排列在矩形格点上。

2. hextop

MATLAB 工具箱的 hextop 函数给出的神经元排列在六方形的格点上，下面举例说明。

【例 12-2】 hextop 函数应用实例。利用 hextop 函数得到一个六边形的包含 6 个神经元的 2×3 的格点阵列。

解： 输入命令：

```
pos = hextop(2,3)
```

得到的输出结果为：

```
pos =   0    1.0000    0.5000    1.5000         0    1.0000
        0         0    0.8660    0.8660    1.7321    1.7321
```

如果我们用 newsom 函数创建一个 SOFM 网络的话，hextop 是默认的网络拓扑形式。

同样，要得到一个包含 80 个神经元的 8×10 的六方形格点阵列并绘图显示，可以输入命令：

```
pos = hextop(8,10);
plotsom(pos)
```

得到的输出结果如图 12-8 所示。

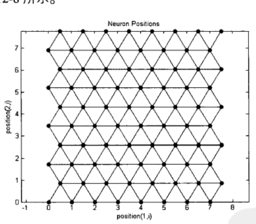


图 12-8 hextop(8,10)生成的格点结构示意的

可以看到，所有神经元确实是排列在六方形格点上。

3. randtop

MATLAB 工具箱的 randtop 函数给出的神经元排列格式为随机格点，下面举例说明。

【例 12-3】 randtop 函数应用实例。利用 randtop 函数得到一个包含 6 个神经元的 2×3 的随机排列的格点阵列。

解： 输入命令：

```
pos = randtop(2,3)
```

得到的输出结果为：

```
pos =     0    0.7620    0.6268    1.4218    0.0663    0.7862
     0.0925       0    0.4984    0.6007    1.1222    1.4228
```

类似的，要得到一个包含 80 个神经元的 8×10 的随机格点阵列并绘图显示，可以输入命令：

```
pos = randtop(8,10);
plotsom(pos)
```

得到的输出结果如图 12-9 所示。

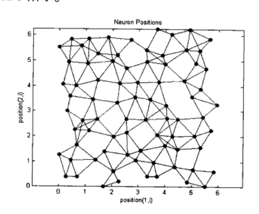


图 12-9 hextop(8,10)生成的格点结构示意图

可以看到，输出的格点排列形式确实是随机无规律的。

下面对 SOFM 网络的距离函数进行介绍。在 MATLAB 工具箱中，有 4 种计算神经元距离的方式，分别对应四个距离函数：dist、linkdist、mandist、boxdist。下面分别对其进行介绍。

1．欧几里得距离

和前面介绍的一样，dist 函数以计算的是两个神经元之间的欧几里得距离。对于两点：

$$x = \{x_1, x_2, \dots, x_n\}$$
$$y = \{y_1, y_2, \dots, y_n\}$$

其欧几里得距离的计算公式如下：

$$Dist = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

【例 12-4】 dist 函数应用实例。定义 3 个神经元格点，并利用 dist 函数得到 3 个神经元的欧几里得距离。

解：输入命令：

```
pos2 = [0 1 2; 0 1 2]
```

输出结果为：

```
pos2 =0     1     2
      0     1     2
```

我们可以通过 dist 函数得到这 3 个神经元两两之间的距离。输入命令：

```
D2 = dist(pos2)
```

得到计算结果为：

```
D2 =      0    1.4142    2.8284
     1.4142         0    1.4142
     2.8284    1.4142         0
```

可以看到，神经元到自己的距离都是 0，而神经元 1 到神经元 2 的距离为 1.414，这正是欧氏空间的直线距离。

图 12-10 显示了 gridtop 格点结构中的一个中心神经元周围的二维邻域的情况。中心神经元的邻域半径逐渐增大，直径为 1 的邻域包含了中心神经元本身以及直接相邻的神经元，而直径为 2 的邻域则包含了直径为 1 的邻近神经元以及与它们直接相邻的神经元。

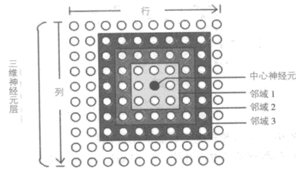


图 12-10　直线距离下二维格点的邻域示意图

对于 dist 函数，一个含有 S 个神经元的神经元层，可以得到一个 $S \times S$ 的距离矩阵。

2. box 距离

box 距离表示两个位置向量之间的距离，假定两个位置向量分别为 $\boldsymbol{P}_i$ 和 $\boldsymbol{P}_j$，则它们之间的 box 距离为：

$$\text{boxdist} = \max(|\boldsymbol{P}_i - \boldsymbol{P}_j|)$$

box 距离对应的 MATLAB 工具箱函数为 boxdist。下面举例说明。

【例 12-5】 boxdist 函数应用实例。定义 6 个神经元格点，并利用 boxdist 函数得到 3 个神经元的 box 距离。

解： 输入命令：

```
pos = gridtop(2,3)
```

输出结果为：

```
pos =0    1    0    1    0    1
     0    0    1    1    2    2
```

应用 boxdist 函数计算各个格点的 box 距离。输入如下命令：

```
d = boxdist(pos)
```

输出结果为：

```
d = 0    1    1    1    2    2
    1    0    1    1    2    2
    1    1    0    1    1    1
    1    1    1    0    1    1
    2    2    1    1    0    1
    2    2    1    1    1    0
```

可以看到，从神经元 1 到神经元 2、3、4 的 box 距离均为 1，因为它们都与神经元 1 直接相邻。而神经元 1 到神经元 5、6 的距离为 2。神经元 3、神经元 4 到其他所有的神经元的距离都为 1。

3. link 距离

两个神经元之间的 link 距离表示的是这两个神经元之间的路径的步数和环节数。应用 linkdist 函数可以求得两向量之间的 link 距离。下面举例说明。

【例 12-6】 linkdist 函数应用实例。用 gridtop 函数定义 6 个神经元格点，并利用 linkdist 函数得到 3 个神经元的 link 距离。

解： 输入命令：

```
pos = gridtop(2,3)
```

输出结果为：

```
pos =0    1    0    1    0    1
     0    0    1    1    2    2
```

应用 linkdist 函数计算各个格点的 link 距离。输入命令：

```
dlink = linkdist(pos)
```

输出结果为：

```
dlink = 0    1    1    2    2    3
        1    0    2    1    3    2
        1    2    0    1    1    2
        2    1    1    0    2    1
        2    3    1    2    0    1
        3    2    2    1    1    0
```



4. manhattan 距离

两个向量 $\boldsymbol{X}$，$\boldsymbol{Y}$ 之间的 manhattan 距离的计算公式如下：

$$D = sum(abs(x-y))$$

x,y 为两个向量中对应的元素，应用 MATLAB 提供的 mandist 函数可以求得两向量之间的 manhattan 距离，需要注意的是，在应用 mandist 函数的时候，两个向量必须一个为行向量，另一个为列向量。下面举例说明。

【例 12-7】 mandist 函数应用实例。定义两个向量，并利用 mandist 函数得到其 manhattan 距离。

解： 输入命令：

```
W1 = [1 2; 3 4; 5 6]
P1 = [1;1]
```

得到的输出为：

```
W1 =1     2
    3     4
    5     6
P1 =1
    1
```

应用 mandist 函数计算向量之间的 manhattan 距离。输入如下命令：

```
Z1 = mandist(W1,P1)
```

输出结果为：

```
Z1 =1
    5
    9
```

可以验证，manhattan 距离的计算是严格按照上面的公式执行的。

以上即 SOFM 网络中常用的四种距离的介绍，下面我们介绍 SOFM 网络结构。

12.2.2 自组织特征映射网络结构

图 12-11 是一个 SOFM 网络的结构图。

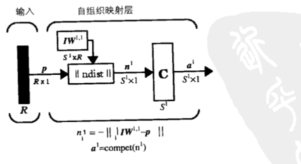


图 12-11　SOFM 网络结构示意图

SOMF 网络结构与自组织竞争网络结构相似，只是前者没有偏差。对于胜出的神经元 i^* 的输出单元 $\boldsymbol{a}^1_i$，竞争传递函数的响应为 1；而对于其他的神经元，竞争传递函数输出均为 0。

如同上面所描述的一样，距离胜出的神经元近的神经元均同时进行权值更新，而网络的格点拓扑结构以及距离形式都是可以选择的。

12.2.3 自组织特征映射网络的学习规则

SOMF 网络的训练过程如下：首先，网络确认获得每一个输入向量对应的胜出神经元，然后定义以获胜神经元为中心的一个邻域，对其中所有神经元的权值矩阵进行调整；随着训练的进行，此邻域将会逐渐缩小，一直到只包含胜出神经元本身为止。

通过这种方式，可以使得网络拓扑结构上位置相近的输出节点的连接权值向量具有一定的相似性同时又有区别，从而保证了对于某一类输入模式，获胜节点能够产生最大的响应，而相邻节点产生的响应较小。SOFM 网络的训练过程与常规的自组织竞争网络的不同之处，就是在于此。

此外，训练过程中，神经元彼此之间相互竞争，具有最大输出的神经元成为胜出者。获胜的神经元节点对其他竞争对手产生抑制的作用，同时对其邻域内的神经元产生激励的作用。只有与胜出节点相邻的神经元的权值才会被修正，并且，此邻域的范围是随着训练的过程不断变化的。

在训练开始时，邻域半径通常较大，调整的节点数量也较多，而随着训练的进行，邻域半径逐渐缩小，最终缩小为仅包括竞争获胜的神经元节点本身。因为只有获胜节点才是输出图形最佳匹配，所以说，SOFM 网络模仿了输入图形的分布状况，提取了输入图形的特征，将其中特征相似的归为一类，对应于某一个节点的胜出。

如上所述，随着训练的进行，决定邻域大小的距离逐渐变化，整个训练可以分为两个阶段。

1. 排序阶段

此阶段的时序步数是事先给定的。在此阶段，邻域距离（邻域半径）首先给定为初始值，然后逐渐减小到调整距离（大小为 1.0）。随着邻域距离的减小，网络神经元根据它们在拓扑结构上的顺序在输入空间上进行排序。

2. 调整阶段

随着邻域距离下降到调整距离，训练进入调整阶段，这个阶段一直持续到训练过程结束。在整个调整阶段，邻域的距离不再下降，始终是调整距离（1.0），在此过程中，网络通过小的邻域进行细调，同时也保持上一阶段的排序结果的稳定。

由此可以看出，特征映射的训练结果是对输入空间进行聚类，同时学习其拓扑结构和分布。

12.3 学习矢量量化网络

为了克服 SOFM 网络作为无监督网络存在的缺点，Kohonen 尝试将竞争学习思想和有监督学习算法结合起来，从而产生了同属于竞争网络的学习矢量量化（Learning Vector

Quantization）网络，简称为 LVQ 网络。这种网络结构给出了分类信息作为导师信号，允许指定将输入分到哪一类。

12.3.1 学习矢量量化网络结构

图 12-12 是一个 LVQ 网络结构的示意图。

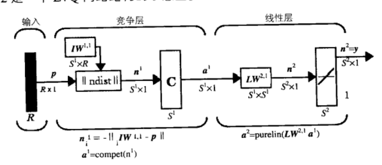


图 12-12 LVQ 网络结构示意图

如图 12-12 所示，LVQ 网络为两层网络结构，第一层为竞争层，第二层为线性层。其中，竞争层对输入向量进行分类的方式类似于 SOFM 网络，线性层将竞争层的分类结果按照用户的定义重新划分为目标类别。竞争层分类的结果被称为子类，而线性层的分类结果成为目标类别。

不论竞争层还是线性层，都是一个神经元对应一个输出类别（既包括子类也包括目标类别）。因此，竞争层能够进行划分的子类数为 S^1，而这 S^1 个子类经过线性层后又会重新组合为 S^2 个目标类别。其中，S^1 总是要大于 S^2。

例如，假定竞争层中的神经元 1、2、3 全部学习了输入空间内从属于线性层目标类别 2 的子类，那么竞争层神经元 1、2、3 与线性层神经元 $\boldsymbol{n}^2$ 的连接权值矩阵 $\boldsymbol{LW}^{2,1}$ 的元素为 1，而与线性层其他神经元的连接权值为 0。如果竞争层中神经元 1、2、3 中的任何一个赢得了竞争，线性层的输入出就为 1。这就是竞争层子类组合为线性层目标类别的方式。

由于我们事先知道竞争层神经元与输出层神经元的子类组合方式，因此就可以事先给出权值矩阵 $\boldsymbol{LW}^{2,1}$。但是，仍然需要通过训练过程以使得第一层神经元对输入向量训练样本集能够产生出正确的子类输出。

12.3.2 学习矢量量化网络的学习规则

学习矢量量化网络采用了有监督的学习规则，包括 LVQ1 与 LVQ2 两种，下面对其进行介绍。

1. LVQ1 学习规则

假设输入、输出样本对为：

$$\{\boldsymbol{p}_1,\boldsymbol{t}_1\},\{\boldsymbol{p}_2,\boldsymbol{t}_2\},\ldots,\{\boldsymbol{p}_N,\boldsymbol{t}_N\}$$

期望输出向量中只有一个元素为 1，其余均为 0，1 代表与输入相关的目标类别。例如，输入和期望输出向量 $\boldsymbol{p}_1$，$\boldsymbol{t}_1$ 为：

$$\boldsymbol{p}_1=\begin{bmatrix}1\\2\\-2\\1\end{bmatrix},\ \boldsymbol{t}_1=\begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix}$$

输入向量中 $\boldsymbol{p}_1$ 包含 4 个元素，期望响应，也就是目标类别，包含 5 个元素，每个输入向量将被划归到 5 个类别之一。初始的分类结果不能保证正确，但网络经过训练后，输入向量将被归到 5 个输出类别中的第 2 类。

训练过程中，输入向量 $\boldsymbol{p}$ 与网络权值矩阵 $\boldsymbol{LW}^{1,1}$ 的行向量之间的距离采用 ndist 函数计算得到。假设网络第一层即竞争层的第 i 个神经元在竞争中胜出，则竞争层传递函数的输出向量 $\boldsymbol{a}^1$ 的第 i 个元素为 1，其余元素均为 0。

在第二层中，利用第一层的输出 $\boldsymbol{a}^1$ 与第二层权值 $\boldsymbol{LW}^{2,1}$ 相乘，结果相当于 $\boldsymbol{a}^1$ 中为 1 的元素选中了相应的类别，这个类别就对应着输入向量的类别 k*，这个分类可能是好的，也可能是不好的，因此，还需要对权值矩阵 $\boldsymbol{LW}^{1,1}$ 的第 i 行调整。

如果分类正确，权值矩阵需要向输入向量 $\boldsymbol{p}$ 移动，如果分类不正确，则向远离输入向量 $\boldsymbol{p}$ 的方向移动。

以公式表示，在分类正确的情况下，有：

$$a_{k^*}^2=t_{k^*}=1$$

从而可以计算权值矩阵 $\boldsymbol{LW}^{1,1}$ 的第 i*行新值：

$${}_{i^*}\boldsymbol{IW}^{1,1}(q)={}_{i^*}\boldsymbol{IW}^{1,1}(q-1)+\alpha(\boldsymbol{p}(q)-{}_{i^*}\boldsymbol{IW}^{1,1}(q-1))$$

如果分类不正确，则：

$$a_{k^*}^2=1\neq t_{k^*}=0$$

此时根据下面的公式对权值进行更新：

$${}_{i^*}\boldsymbol{IW}^{1,1}(q)={}_{i^*}\boldsymbol{IW}^{1,1}(q-1)-\alpha(\boldsymbol{p}(q)-{}_{i^*}\boldsymbol{IW}^{1,1}(q-1))$$

可以通过误差反向传播的方式对权值矩阵 $\boldsymbol{IW}^{1,1}$ 第 i*行进行修正，而不对其他行产生影响。以上的修正公式使得竞争层神经元朝着目标类别的方向移动，最终落入正确的分类空间，同时远离其他类别向量。

MATLAB 中 LVQ 网络学习规则对应的函数为 learn1v1。

2. LVQ2 学习规则

LVQ2 学习规则可以应用在 LVQ1 之后，从而提高其第一步训练的效果。LVQ2 学习规则与 LVQ1 学习规则类似，不同之处在于 LVQ2 规则中第一层中只有与输入向量最接近

的两个向量得到更新，其中一个对应于正确的分类，而另一个对应于错误的类别，这样使得输入向量落入这两个向量的中位面的一个“窗口”中。

窗口的定义如下：

$$\min(\frac{d_i}{d_j},\frac{d_j}{d_i}) > s$$

其中 $s=\frac{1-\omega}{1+\omega}$，而 d_i，d_j 分别为输入向量 $\boldsymbol{p}$ 到权值矩阵向量 ${}_{i^*}\boldsymbol{IW}^{1,1}$ 和 ${}_{j^*}\boldsymbol{IW}^{1,1}$ 的欧氏距离。假定 ω 的变动范围为 0.2 到 0.3。例如，如果选择 ω=0.25，则 s=0.6。这表示如果两者的距离比值超过 0.6，这两个向量就要进行调整。

也就是说，如果输入向量 $\boldsymbol{p}$ 与 ${}_{j^*}\boldsymbol{IW}^{1,1}$ 属于同一类，而 $\boldsymbol{p}$ 与 ${}_{i^*}\boldsymbol{IW}^{1,1}$ 则不属于同一类，而输入又在中位面附近，则需要对两个向量进行调整。

权值的调整公式为：

$${}_{i^*}\boldsymbol{IW}^{1,1}(q) = {}_{i^*}\boldsymbol{IW}^{1,1}(q-1) - \alpha(\boldsymbol{p}(q) - {}_{i^*}\boldsymbol{IW}^{1,1}(q-1))$$

$${}_{j^*}\boldsymbol{IW}^{1,1}(q) = {}_{j^*}\boldsymbol{IW}^{1,1}(q-1) + \alpha(\boldsymbol{p}(q) - {}_{j^*}\boldsymbol{IW}^{1,1}(q-1))$$

因此，如果两个权值向量与输入最接近，一个属于错误的类别而另一个属于正确的类别，而输入又落入中位面窗口之内，则这两个向量要进行调整。这样的一个训练过程可以使得经过 LVQ1 训练初步分类处理的权值向量进行更细致的调整，得到的训练结果更加稳健。

MATLAB 中 LVQ2 学习规则对应的函数为 learn1v2。

12.3.3 与自组织映射网络的比较

LVQ 网络与 SOFM 网络都是芬兰学者 Kohonen 提出的网络模型，同属于自组织竞争型网络，两者在网络结构上非常相似。

SOFM 网络的最大优点是在网络输出层引入了拓扑结构，从而实现了对生物神经网络竞争过程的模拟；而 LVQ 网络则在竞争学习的基础上引入了有监督学习算法，被认为是 SOFM 网络的扩展形式。

常用的结合方法是，将学习矢量量化算法作为自组织映射算法的补充，在输出层应用具有拓扑结构的自组织映射网络结构，依次采用自组织映射学习算法和学习矢量量化算法对网络进行两次训练。

在网络训练过程中，首先采用自组织映射算法进行训练，网络以无监督的方式对输入进行特征提取，目标是选择一个小的合理的特征集合，使它包含输入数据的主要特征信息。然后在此基础上，采用学习矢量量化学习算法对网络进行二次训练。在原先训练的基础上指定网络输出神经元所属的类别。其中，二次训练的过程才是对网络进行实际分类的过程。

实践证明，在加入学习矢量量化后，网络的分类边界能够进一步地收缩，从而实现对输入向量更为准确的划分。

总之，LVQ 网络作为一种结合了自组织特征映射与有监督学习算法的混合网络结构，

本身输出层并不具有拓扑结构，对于网络的竞争学习并不是对生物神经系统的最好模拟，但是应用在实际系统中，却可以提高系统的整体性能。

12.4 自组织神经网络的 MATLAB 实现

12.4.1 自组织竞争网络的设计

利用 MATLAB 工具箱函数 newc 可以创建一个竞争型神经网络，其调用格式为：

newc = newc(P, S, KLR, CLR)

其中：

- P 为输入范围；
- S 为神经元个数；
- KLR 为 Kohonen 学习速率，默认值为 0.01；
- CLR 为 Conscience 学习速率，默认值为 0.001。

下面通过一个简单的示例来说明它是如何工作的。

【例 12-8】 定义二维输入向量 $\boldsymbol{p}$ = [.1, .8, .1, .9; .2, .9, .1, .8]，创建一个竞争型网络对其进行模式划分。

解： ① 定义输入向量。输入如下命令：

```
p = [.1 .8 .1 .9; .2 .9 .1 .8]
```

输出结果为：

```
p =0.1000    0.8000    0.1000    0.9000
     0.2000    0.9000    0.1000    0.8000
```

其中有两个向量接近于原点(0,0)，两个接近于(1,1)。

② 创建一个两神经元的网络，输入变动范围为[0,1]，调用函数 newc 创建竞争型神经网络。输入命令：

```
net = newc([0 1; 0 1],2);
```

生成的网络权值位于输入范围的中点位置，可以利用下面的命令查看：

```
wts = net.IW{1,1}
```

输出结果为：

```
wts =0.5000    0.5000
       0.5000    0.5000
```

可以看到，如我们所期待的一样，网络的初始权值确实位于输入范围的中值位置。偏差值则是用函数 initcon 计算得到的。输入下面语句查看偏差：

```
biases = net.b{1}
```

给出的结果为：

```
biases = 5.4366
         5.4366
```

这样我们就得到了一个竞争型神经网络，但是我们仍然需要对其进行训练，使其能够正确地完成分类的工作。注意，训练过程中，每一个神经元需要竞争以对输入向量 $\boldsymbol{p}$ 进行响应。

如果偏差值为 0，就是最接近输入向量 $\boldsymbol{p}$ 的那个神经元传递函数获得的输入最大，并且产生的输出为 1，其他所有的神经元输出为 0。关于自组织竞争网络的训练我们将在下面进行介绍。

12.4.2 自组织竞争网络的训练

【例 12-9】 自组织竞争网络的训练实例。对【例 12-8】的网络进行训练，使其能够对输入向量正确地分类。

解： ① 接上例运行的结果，首先查看一下利用 newc 函数生成的竞争网络的默认网络训练函数。输入命令：

```
net.trainFcn
```

输出结果为：

```
ans =trainr
```

这表明网络的初始训练函数为 trainr。我们将网络的训练迭代次数设置为 500，并使用 train 函数进行训练。输入命令：

```
net.trainParam.epochs = 500;
net = train(net,p);
```

对每一次迭代，所有的训练样本向量都按随机的顺序输入给网络。而权值与偏差都在每一次样本向量输入后进行修正。

② 利用输入向量对训练后的网络进行仿真，最后将输出向量转化为标量形式显示。输入如下命令：

```
a = sim(net,p)
ac = vec2ind(a)
```

得到结果为：

```
ac =     1     2     1     2
```

可以看到，网络经过训练后，将输入向量划分到了两个目标类别中，其中接近原点(0,0)的输入向量划分为类别 1，而接近(1,1)的输入向量，划分到类别 2。

对经过训练后的网络，我们可以再次查看其权值矩阵和偏差向量，得到的结果为：

```
wts = 0.1000    0.1467
      0.8474    0.8525
```

```
biases = 5.4961
         5.3783
```

在实际运行过程中，由于输入样本向量的顺序具有一定的随机性，查看权值偏差所得到的实际数值可能有所不同。

注意，第一个权值向量，也就是权值矩阵的第一行构成的向量接近于输入向量中的原点；而第二个权值向量，也就是权值矩阵的第二行构成的向量，则接近于输入向量中的(1,1)，因此，网络经过训练之后就能够对输入的向量进行正确的分类。

在训练过程中，竞争层每一个靠近输入向量类别的神经元权值矩阵都会朝着输入向量的方向移动。最终，如果神经元数目足够多的话，每一组相似的输入向量将会对应一个神经元产生等于 1 的输出，而对于其他神经元产生的输出均为 0，从而实现对输入向量的识别。

③ 通过图形显示的方式，竞争层的权值向量和输入向量之间的关系可以得到更好的表现。下面的图 12-13 以加号“+”显示了一个包含两个元素的输入向量。

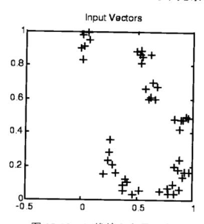


图 12-13 二维输入向量显示意图

上面的输入向量呈现出成组的方式，这样的情形利用竞争型网络进行模式划分是非常合适的。

12.4.3 SOFM 网络的设计

应用 MATLAB 工具箱函数 newsom 可以快速地创建一个自组织特征映射类型的神经网络。其调用格式如下：

net = newsom(***P***, [d1,d2,...], tfcn, dfcn, steps, in)

其中：

- ***P*** 是 $R \times Q$ 阶矩阵，包含 Q 个输入向量；
- di 是第 i 层的大小，默认值为[5 8]；
- tfcn 是网络的拓扑函数，默认值为 hextop；
- dfcn 是网络的距离函数，默认值为 linkdist；

- steps 是调整阶段邻域变为 1 的步骤，默认值为 100；
- in 是初始网络大小，默认值为 3。

以下面的实例进行说明。

【例 12-10】 SOFM 网络设计实例。假定输入向量为 2 维，取值区间分别为[0，2]和[0，1]，以此来建立一个 SOFM 网络，此网络包含 6 个神经元，按 2×3 的形式排列。

解：① 利用 newsom 函数生成 SOFM 网络。输入如下命令：

```
net = newsom([0 2; 0 1],[2 3]);
```

② 定义输入向量。输入命令：

```
P = [.1 .3 1.2 1.1 1.8 1.7 .1 .3 1.2 1.1 1.8 1.7;...
0.2 0.1 0.3 0.1 0.3 0.2 1.8 1.8 1.9 1.9 1.7 1.8]
```

③ 通过 plotsom 函数绘制出生成的网络的拓扑结构图。输入命令：

```
plot(P(1,:),P(2,:),'o','markersize',8)
hold on
plotsom(net.iw{1,1},net.layers{1}.distances)
hold off
```

得到的图形如图 12-14 所示。

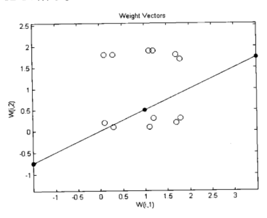


图 12-14　网络拓扑结构

其中的空心圆点代表网络的输入向量，实心圆点代表网络的初始权值向量。可以看到，输入向量分布在图像的中间偏上的位置，而生成的网络权值向量初始分布在左、中、右三个点处。可以对输出权值向量进行查看，输入命令：

```
net.iw{1,1}
```

得到输出为：

```
ans =3.5000    1.7500
     3.5000    1.7500
     1.0000    0.5000
```

```
    1.0000    0.5000
   -1.5000   -0.7500
   -1.5000   -0.7500
```

可以看到，网络权值确实集中在三个点。对网络进行仿真时，计算每个神经元权值向量相对于输入向量的负距离，从而得到权值输入，也即特征映射层传递函数的输入，然后再通过竞争训练从而确定胜出的神经元，并产生 1 的响应输出。

12.4.4 SOFM 网络的训练

在默认的情况下，应用 newsom 生成的 SOFM 网络的学习模式为批处理模式 trainbuwb，而权值向量的训练函数为 learnsomb。下面进行举例说明。

【例 12-11】 SOFM 网络训练实例。针对【例 12-10】生成的网络进行训练，并查看结果。

解： ① 直接利用 train 函数对网络进行训练，设定最大训练迭代次数为 1000。输入命令：

```
net.trainParam.epochs = 1000;
net = train(net,P);
```

可以从弹出的 nntraintool 窗口中看到，训练过程一直持续到 1000 次迭代后结束。通过 plotsom 函数绘制出训练后网络的拓扑结构图。输入如下命令：

```
plotsom(net.iw{1,1},net.layers{1}.distances)
```

绘出图形如图 12-15 所示。

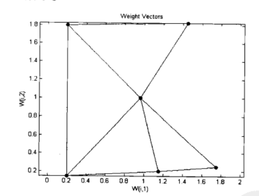


图 12-15 训练后网络的拓扑结构

可以看到，经过训练后，SOFM 网络中各个神经元的权值都发生了移动。如果继续训练，各个神经元的权值都会朝着输入向量的方向移动，从而学习输入空间的拓扑结构，对输入向量进行模式划分。

12.4.5 LVQ 网络的设计

应用 MATLAB 工具箱函数 newlvq 函数可以创建一个 LVQ 网络，其调用格式如下：

net = newlvq(pr, s1, pc, lr, lf)

其中：

- pr 为 $R\times2$ 矩阵，代表输入向量元素的最小和最大值；
- $s1$ 为隐层的神经元数目；
- pc 为二元向量，表示典型的类别权重百分比；
- lr 为学习速率，默认值为 0.01；
- lf 为学习函数，默认值为 learn1v1。

下面举例说明 newlvq 函数的用法。

【例 12-12】 LVQ 网络设计实例。假定有 10 个二维输入向量，以此来建立一个 LVQ 网络，将其划分为 4 个子类，此网络竞争层包含 4 个神经元，然后通过线性输出层，将其划归到两个输出目标类别。

解： ① 定义输入样本和对应的目标类别。输入如下命令：

```
P = [-3 -2 -2 0 0 0 0 2 2 3; 0 1 -1 2 1 -1 -2 1 -1 0],
Tc = [1 1 1 2 2 2 2 1 1 1],
```

得到输出结果为：

```
P = -3    -2    -2    0    0    0    0    2    2    3
     0     1    -1    2    1   -1   -2    1   -1    0
Tc =1     1     1    2    2    2    2    1    1    1
```

可以利用 plotvec 函数对不同类别的输入向量用不同的颜色进行显示。输入命令：

```
plotvec(P,Tc,'*'),
```

得到的绘图结果如图 12-16 所示。

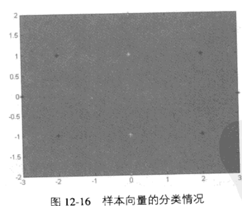


图 12-16 样本向量的分类情况

其中黑色点代表被划分为目标类别 1 的点，灰色点代表被划分为目标类别 2 的点。生成的网络首先要将输入类别划分为 4 个子类，而最终目标是要将向量 $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_8, \boldsymbol{p}_9$，以及 $\boldsymbol{p}_{10}$ 划分到目标类别 1，而向量 $\boldsymbol{p}_4, \boldsymbol{p}_5, \boldsymbol{p}_6$，以及 $\boldsymbol{p}_7$ 划分到目标类别 2 中。

注意，因为这个问题不是线性可分的，因此用感知器神经网络无法解决，但是 LVQ

网络可以很容易地解决这个问题。

② 将目标类别 Tc 转化为 LVQ 网络所需的目标向量的形式。输入如下命令：

```
T = ind2vec(Tc);
```

得到的结果是一个稀疏矩阵 $\boldsymbol{T}$，可以用如下命令进行调整，使之转化为一个满存储矩阵，以便于查看：

```
targets = full(T)
```

输出结果为：

```
targets = 1    1    1    0    0    0    0    1    1    1
          0    0    0    1    1    1    1    0    0    0
```

这个结果看来 targets 矩阵正确地给出了各个输入向量与目标类别的关系。下面利用 newlvq 来创建 LVQ 网络。

需要创建的网络其第一层有四个神经元，第二层有两个神经元。第一层的权值被初始化为输入向量范围的中值，这是由 midpoint 函数确定的；第二层的权值中有 60%对应着 targets 第一行中的元素 1，40%对应着 targets 中第二行元素 1。输入命令：

```
net = newlvq(P,4,[.6 .4]);
```

查看竞争层初始权值。输入命令：

```
net.IW{1,1}
```

输出结果为：

```
ans =0     0
     0     0
     0     0
     0     0
```

可以看到，初始网络权值全都为 0，这确实是输入向量的范围–3 到 3 的中点。同样可以查看第二层的权值。输入命令：

```
net.LW{2,1}
```

输出结果为：

```
ans =1     1     0     0
     0     0     1     1
```

这个结果同样也是可以理解的，它的含义是：如果竞争层输出为 1，则被划归为目标类别 1；如果竞争层输出为 0，则被划归为目标类别 2。也就是说，竞争层的前两个神经元连接着线性层的第一个神经元，竞争层的后两个神经元连接着线性层的第二个神经元。

③ 利用 sim 函数对网络进行仿真。输入命令：

```
Y = sim(net,P);
Yc = vec2ind(Y)
```

输出结果为：

```
Yc = 1    1    1    1    1    1    1    1    1    1
```

输出结果说明所有的输入向量都被划入了类别 1，这不是我们期望的结果，因为我们还没有对网络进行训练。训练过程将对第一层的权值进行调整，从而得到期望的分类结果，训练函数将在下一节介绍。

12.4.6 LVQ 网络的训练

【例 12-13】 LVQ 网络训练实例。对【例 12-12】生成的网络进行训练，并查看结果。

解： ① 对网络进行训练的目的是调整第一层的权值，从而使之生成正确的分类结果。首先将训练迭代次数设置为 150，然后调用 train 函数。输入如下命令：

```
net.trainParam.epochs = 150;
net = train(net,P,T);
```

然后查看第一层的权值，输入命令：

```
net.IW{1,1}
```

输出结果为：

```
ans =0.2861    0.0095
  -0.1097    0.0000
        0   -0.0100
        0         0
```

由于训练过程的随机性，每次训练的权值结果并不一定完全相等。输入下面的命令，对其进行绘图显示：

```
cla;
plotvec(P,Tc,'*'),
hold on;
plotvec(net.IW{1}',vec2ind(net.LW{2}),'o');
hold off;
```

绘出图形如图 12-17 所示。

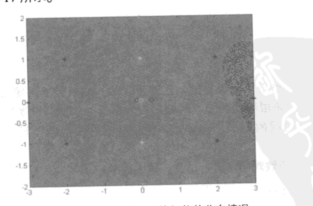


图 12-17　训练后竞争层网络权值的分布情况

图中不同点的颜色深浅代表不同的类别，星型点代表输入和目标类别样本点，空心圆圈代表为训练后的网络权值及其对应目标类别。可以看到，竞争层网络权值朝着输入向量的方向发生了移动。

利用 sim 函数仿真，验证网络训练结果。输入命令：

```
Y = sim(net,P);
Yc = vec2ind(Y)
```

输出结果为：

```
Yc = 1    1    1    2    2    2    2    1    1    1
```

可以看到，经过训练的网络确实能够正确地对输入向量进行分类。

② 作为最后一步检查，我们可以利用一个与样本向量接近的输入向量进行测试。输入命令：

```
pchk1 = [0; 0.5];
Y = sim(net,pchk1);
Yc1 = vec2ind(Y)
```

输出结果为：

```
Yc1 =   2
```

结果是正确的，因为 pchk1 向量与输入向量中属于类别 2 的部分更接近。类似的，我们可以试验另一个输入向量。输入如下命令：

```
pchk2 = [1; 0];
Y = sim(net,pchk2);
Yc2 = vec2ind(Y)
```

输出结果为：

```
Yc2 =    1
```

结果同样是正确的，因为 pchk2 向量与输入向量中属于类别 1 的部分更接近。

12.5 自组织神经网络应用实例

本章介绍的各种类型的自组织神经网络应用方向主要集中在模式分类领域，这一节我们利用实例对其进行介绍。

12.5.1 自组织竞争网络模式分类

【例 12-14】 自组织竞争神经网络模式分类应用实例。对给定的具有一定分类特征的随机输入向量 $\boldsymbol{p}$，创建自组织竞争神经网络对其进行分类。

解： ① 定义具有一定随机性和分类结构的输入向量。输入如下命令：

```
X = [0 1; 0 1];        % 定义输入类别范围
```

```
clusters = 8;          % 类别数目
points = 10;           % 每一类中的样本数
std_dev = 0.05;        % 每一类的标准差
P = nngenc(X,clusters,points,std_dev);     %生成输入样本向量序列
```

其中 nngenc 是 MATLAB 神经网络工具箱中用来生成一系列输入样本向量的函数，该函数需要的 4 个参数分别为：输入向量类别范围、类别数目、每一类中的样本数，以及每一类的标准差。下面对输入空间进行绘图显示。输入命令：

```
plot(P(1,:),P(2,:),'+r');
title('Input Vectors');
xlabel('p(1)');
ylabel('p(2)');
```

绘出的输入空间向量分布如图 12-18 所示。

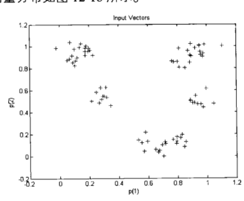


图 12-18　输入向量空间分布

可以看到，随机输入向量在空间成团地分布，大约分为 8 组。

② 下面利用 newc 函数创建自组织竞争网络，给定参数分别为：输入空间的最小和最大值、神经元数目以及学习速率。然后，将其权值向量绘图显示，对应其初始分类结果。输入如下命令：

```
net = newc([0 1;0 1],8,.1);
w = net.IW{1};
plot(P(1,:),P(2,:),'+r');
hold on;
circles = plot(w(:,1),w(:,2),'ob');
```

绘图结果如图 12-19 所示。

图中圆圈表示的就是初始的网络权值向量位置。可以看到，网络的初始权值位于[0.5, 0.5]处，并不能够很好地体现输入向量的空间分布情况。接下来对设计的竞争网络进行训练，设定最大迭代次数为 7 次，然后对训练后的网络权值进行绘图显示。输入命令：

```
net.trainParam.epochs = 7;
net = train(net,P);
w = net.IW{1};
```

```
delete(circles);
plot(w(:,1),w(:,2),'ob');
```

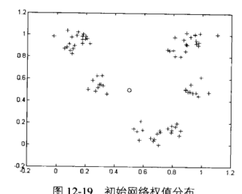


图 12-19 初始网络权值分布

绘出图形如图 12-20 所示。

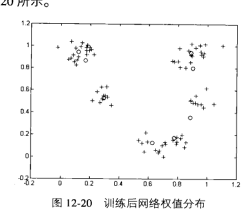


图 12-20 训练后网络权值分布

此时即可以应用此竞争网络进行模式分类，其中每一个神经元对应一个不同的输入类别。设定一个输入向量 $\boldsymbol{p}$ 对网络进行仿真。输入命令：

```
p = [0; 0.2];
a = sim(net,p)
```

输出结果为：

```
a =   (2,1)        1
```

函数执行的结果给出了对此输入向量响应的神经元，也就是输入向量对应的类别。

12.5.2 一维自组织特征映射网络

自组织网络可以表示输入空间中输入向量的位置，通过相邻神经元对相似的输入产生响应，使得网络层可以学习并表示出输入向量空间的拓扑结构。

【例 12-15】 一维自组织特征映射网络应用实例。设计一个一维的 SOFM 网络，对

一维输入空间的拓扑结构进行表示。

解： ① 定义 100 个输入向量，均位于输入空间的单位圆上，对其进行绘图显示。输入如下命令：

```
angles = 0:0.5*pi/99:0.5*pi;
P = [sin(angles); cos(angles)];
plot(P(1,:),P(2,:),'+r')
```

得到输入向量的空间分布图如图 12-21 所示。

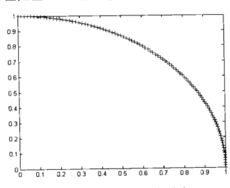


图 12-21　输入向量空间分布

利用 newsom 函数创建一个一维的 SOFM 网络，包含 10 个神经元。其中第一个参数代表网络有两个输入向量，其变化范围均为[0 1]。

```
net = newsom([0 1;0 1],[10]);
```

② 定义网络训练的最大迭代次数为 10 次，利用 train 函数对其进行训练。输入命令：

```
net.trainParam.epochs = 10;
net = train(net,P);
```

然后将训练后网络权值进行绘图显示。输入命令：

```
plotsom(net.iw{1,1},net.layers{1}.distances)
```

绘出图形如图 12-22 所示。

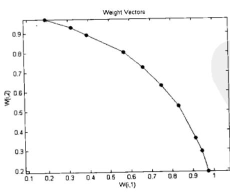


图 12-22　训练后的网络权值分布

观察图 12-21 与图 12-22 的相似性，可以看到，此时的网络权值正是输入空间的拓扑结构的映射。当输入样本向量时，网络中神经元 1 到 10 中至少有一个的输出为 1，代表这个输入向量所属的空间划分类别。我们可以用其中一个向量进行仿真，输入命令：

```
p = [1;0];
a = sim(net,p)
```

输出结果为：

```
a =  (10,1)        1
```

这表示了输入向量 $\boldsymbol{p}$ 有响应的神经元，也即 $\boldsymbol{p}$ 所属的类别。

12.5.3 二维自组织特征映射网络

对于具有一定特征分布的二维输入样本向量，可以利用二维 SOFM 网络对其进行分类。下面举例进行说明。

【例 12-16】 二维 SOFM 网络应用实例。设计一个二维的 SOFM 网络，对 1000 个二维样本输入向量进行分类。

解： ① 定义 1000 个矩形输入空间内的二维随机输入向量，对其进行绘图显示。输入如下命令：

```
P = rands(2,1000);
plot(P(1,:),P(2,:),'+r')
```

绘出的图形如图 12-23 所示。

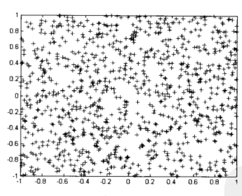


图 12-23 输入向量的空间分布

② 由于输入空间的向量分布与拓扑结构比较复杂，需要用一个复杂度比较高的网络来进行划分。这里设计一个 6 层的神经网络，每层含 5 个神经元。其中每个神经元代表一类划分，也即矩形输入空间的不同区域，相邻的神经元对应着相邻的空间区域。

利用 newsom 函数设计网络。输入命令：

```
net = newsom([0 1; 0 1],[5 6]);
```

然后利用函数 plotsom 对生成的网络权值进行绘图显示。输入命令：

```
plotsom(net.iw{1,1},net.layers{1}.distances)
```

绘出图形如图 12-24 所示。

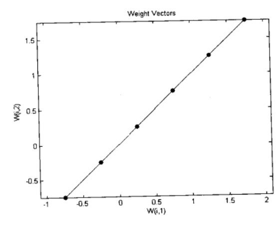


图 12-24　初始生成的网络权值分布

图中圆点代表网络的权值向量。可以看到，初始生成的网络权值的各层初始值是相同的，并且都位于同一直线上，因此图中只能看到 5 个圆点。

③ 下面利用①中生成的 1000 个输入向量，对网络进行训练，并重新绘制网络权值图。输入命令：

```
net.trainParam.epochs = 1;
net = train(net,P);
plotsom(net.iw{1,1},net.layers{1}.distances)
```

绘出图形如图 12-25 所示。

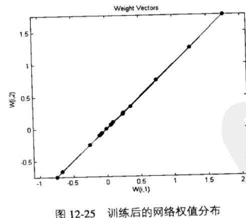


图 12-25　训练后的网络权值分布

可以看到，经过一步迭代的训练，网络权值移动了，每一个权值代表着输入空间的不同区域，以此对输入空间进行分类。

④ 下面利用 sim 函数对输入向量进行仿真。输入命令：

```
p = [0.5;0.3];
a = sim(net,p)
```

输出结果为：

```
a =  (14,1)        1
```

可以看到，对此输出响应为 1 的神经元为(14,1)，也就是此输入向量对应的类别。

12.5.4 LVQ 网络应用实例

【例 12-17】 LVQ 网络应用实例。设计一个 LVQ 网络，根据期望目标向量对输入向量进行分类。

解：① 定义一个 10 个二维输入样本向量 $\boldsymbol{P}$，以及期望目标类别 $\boldsymbol{C}$，并将 $\boldsymbol{C}$ 转化为向量形式 $\boldsymbol{T}$。输入命令：

```
P = [-3 -2 -2  0  0  0  0 +2 +2 +3;
      0 +1 -1 +2 +1 -1 -2 +1 -1  0];
C = [1 1 1 2 2 2 2 1 1 1];
T = ind2vec(C);
```

对输入数据点绘图显示。输入命令：

```
cla      %清除图形
for i=1:10
if(C(i)==1)      %对第一类输入以加号“+”显示
plot(P(1,i),P(2,i),'+')
hold on;
else
plot(P(1,i),P(2,i),'o')           %对第二类输入以空心圆圈“o”显示
hold on;
end
end
```

绘出图形如图 12-26 所示。

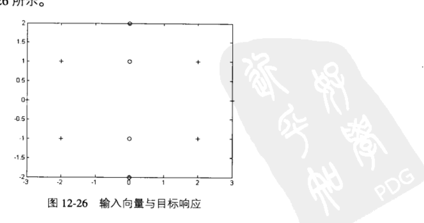


图 12-26 输入向量与目标响应

其中，第一类输入向量用加号“+”表示，第二类输入向量用空心圆圈“o”表示。

(2) 利用函数 newlvq 设计一个 LVQ 网络，对上述数据点进行分类。这里需要输入 4 个参数：输入向量的最小和最大值、隐层神经元数目、表示典型类别的百分比的向量，以及学习速率。输入命令：

```
net = newlvq(minmax(P),4,[.6 .4],0.1);
```

对 LVQ 网络进行绘图显示。输入命令：

```
hold on
W1 = net.IW{1};
plot(W1(1,1),W1(1,2),'*')
title('Input/Weight Vectors');
xlabel('P(1), W(1)');
ylabel('P(2), W(3)');
```

绘出图形如图 12-27 所示。

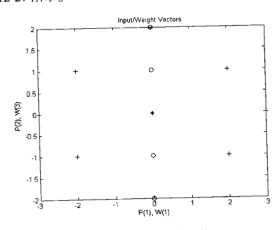


图 12-27　初始网络权值

图中，初始网络权值向量以星号“*”表示，初始生成的网络权值都在(0,0)位置。

(3) 对网络进行训练，设定最大迭代次数为 150，并重新绘图显示。输入命令：

```
net.trainParam.epochs=150;
net=train(net,P,T);

W1 = net.IW{1};
W2 = vec2ind(net.IW{1});

cla;          %清除图形
for i=1:10
    if(C(i)==1)
        plot(P(1,i),P(2,i),'+') %对第一类输入向量以加号 "+" 显示
        hold on;
    else
        plot(P(1,i),P(2,i),'o') %对第二类输入向量以空心圆圈 "o" 显示
```

```
        hold on;
    end
end
for i=1:4
    if(W2(i)==1)          %对第一类神经元以星号“*”显示
        plot(W1(i, 1),W1(i, 2),'*')
        hold on;
    else                  %否则，以实心圆点“.”显示
        plot(W1(i, 1),W1(i, 2),'.')
        hold on;
    end
end
```

绘出图形如图 12-28 所示。

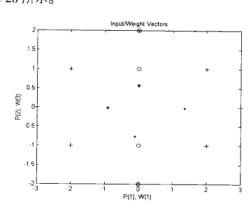


图 7-28 训练后的网络权值与输入向量

图中，LVQ 网络神经元中的第一类以星号“*”表示，第二类以实心圆点“.”表示。

(4) 下面对此网络进行仿真，输入向量为[0.2; 1]。输入命令：

```
p = [0.2; 1];
a = vec2ind(sim(net,p))
```

输出结果为：

```
a =     2
```

此结果说明训练后的 LVQ 网络将此输入向量归入了第二类中。

12.6 小结

本章分别对自组织竞争神经网络、SQM 网络、LVQ 网络的结构及其训练算法进行了介绍，并通过实例介绍了其应用。

第 13 章　反馈神经网络

1982 年，美国物理学家 Hopfield 发表了一篇对人工神经网络很有影响的论文。在论文中，他提出了一种具有循环互联的反馈型人工神经网络模型，引入了“能量函数”的概念，并给出了网络的稳定性判据。以此为基础发展出的神经网络模型被称为反馈型神经网络（Recurrent Network），又称为递归神经网络或回归网络。

反馈神经网络的目标是设计一个能够存储一组平衡点的网络，使得当给定一组网络初始值时，网络能够通过自运行最终收敛达到这些设计的平衡点。

Hopfield 将反馈神经网络应用于约束优化问题，如 TSP 问题的求解、实现 A/D 转换等，又利用网络的吸引子及其吸引域实现了信息的联想记忆功能。此外，由于 Hopfield 网络与电子模拟线路之间存在着很好的对应关系，非常易于理解和实现，因此也促进了人工智能和神经网络计算机的发展。

反馈神经网络的系统平衡状态可以通过设计网络的权值而存储到网络中，并且一个网络可以有多个稳定状态，当系统从某一初始状态开始运动时，总是可以收敛到一个稳定的平衡状态。因此反馈神经网络属于动态类型的网络。

如果将网络平衡状态看做网络的记忆，那么网络由任意的初始状态向稳态变化的过程就可以看做网络的一种寻找记忆的过程。反馈神经网络具有联想记忆的能力，而网络的稳定平衡点正是网络联想记忆的基础。

反馈神经网络与前面所介绍的前馈型神经网络的不同点在于，反馈神经网络更加关心网络的平衡点和稳定性，即如何得到和利用稳定的网络，而前馈神经网络更加关注学习和训练过程，以及非线性处理性能。

13.1 Hopfield 网络

Hopfield 网络最具有吸引力的性能是可以实现联想记忆的功能，其网络结构如图 13-1 所示。

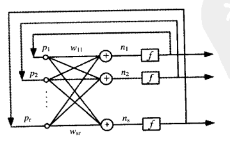


图 13-1　Hopfield 网络结构示意

图中网络具有 r 个输入，s 个神经元，为单层的全反馈结构。每个神经元的输出又与其他神经元的输入互连，通过连接权值构成神经元的输入，在这种结构下必然有 $r=s$，即网络输入的数目与输入层神经元的数目是相等的。

按照传递函数的不同，还可以分为离散 Hopfield 网络（Discrete Hopfield Network，简称 DHNN）和连续 Hopfield 网络（Continuous Hopfield Network，简称 CHNN）。Hopfield 网络的应用范围很广，包括图像处理、语音处理、数据查询、容错计算、模式识别、模式分类等方向，本节对两类 Hopfield 神经网络进行介绍。

13.1.1 离散 Hopfield 网络模型

Hopfield 最早提出的反馈网络采用二值传递函数，神经元的输出只有 0 或 1 两个值，对应于图 13-1 中的传递函数 f 为硬限值函数，因此称为离散 Hopfield 网络。离散 Hopfield 网络中，神经元的输出为 0 或 1，分别表示神经元处于抑制或者激活状态。

假设 Hopfield 中包含 n 个神经元，从神经元 j 输出到神经元 i 输入的互连权值表示为 w_{ij}，任意神经元 i 在 t 时刻传递函数的输入表示为 $u_i(t)$，t 时刻的输出表示为 $v_i(t)$，偏差值为 b_i，则 t 时刻的传递函数输入可以表示为：

$$u_i(t)=\sum_{j=1}^{n} w_{ij}v_j(t)+b_i$$

对应神经元的输出状态为：

$$v_i(t+1)=f\left(u_i(t)\right)$$

其中传递函数 f 可以取阶跃函数或者符号函数。如果取符号函数，Hopfield 网络的神经元输出 $v_i(t+1)$ 取离散值 1 或者−1，即：

$$v_i(t+1)=\begin{cases}1, & \sum_{j=1}^{n} w_{ij}v_j(t)+b_i \geqslant 0 \\ -1, & \sum_{j=1}^{n} w_{ij}v_j(t)+b_i \leqslant 0\end{cases}$$

Hopfield 网络按照动力学规则运行，其工作过程就是状态的演化过程，从初始状态按照李雅普诺夫能量函数减小的方向演化，直到达到稳定状态，得到稳定的网络输出。Hopfield 网络的运行包括串行和并行两种工作模式。

（1）串行工作模式。在任意时刻 t，只有某一个神经元 i 的状态发生变化，而其他神经元的状态保持不变。这种工作模式也称为异步模式。

（2）并行工作模式。在任意时刻 t，可以有多个神经元的状态同时发生变化，这种工作模式也称为同步模式。

下面以串行工作模式为例说明 Hopfield 网络的运行步骤。

① 对网络进行初始化。

② 从网络中随机选取一个神经元。

③ 按照公式计算出神经元的输入 $u_i(t)$。

④ 按照公式求出该神经元的输出 $v_i(t+1)$，此时网络中的其他神经元输出保持不变。

⑤ 判断网络是否达到稳定状态，若达到稳定状态或者满足给定的条件，则结束训练，否则转到第②步继续执行。

反馈网络的稳定性是一个非常重要的指标。若网络从某一时刻后，状态不再发生变化，就称网络达到了稳定状态。此时，下一时刻的网络输出与当前时刻的网络输出相等，用公式表示为：

$$v_i(t+1) = v_i(t)$$

对于串行工作模式的网络，其稳定性称为串行稳定性，对于并行工作的网络，其稳定性称为并行稳定性。

1983 年，Coben 和 Grossberg 给出了关于 Hopfield 网络稳定的充分条件。要使得 Hopfield 网络存在稳定状态，只需网络为对称连接，即满足：

$$w_{ij} = w_{ji}$$

且神经元自身无连接，即：

$$w_{ii} = 0$$

因此，Hopfield 网络的连接权值矩阵通常设为零对角对称矩阵。不过，这只是 Hopfield 网络稳定的充分条件而不是必要条件，因此实际中也有很多稳定的 Hopfield 网络并不满足连接权值矩阵是零对角对称矩阵这一条件。连接权值矩阵具有对称性的 Hopfield 网络如图 13-2 所示。

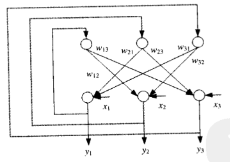


图 13-2　零对角对称权值矩阵的 Hopfield 网络示意图

由于具有反馈，Hopfield 网络是一个非线性动力学系统，网络按照非线性动力学的方式运行，即按照“能量函数”减小的方向进行演化，直到达到稳定状态。其网络的“能量函数”为李雅普诺夫函数，定义为：

$$E = -\frac{1}{2}\sum_{i=1,i\neq j}^{n}\sum_{j=1,j\neq i}^{n} w_{ij}v_i v_j + \sum_{i=1}^{n} b_i v_i$$

其中 w_{ij} 为 i，j 神经元的互联权值，在满足以上参数条件下，李雅普诺夫函数的能量在网络运行过程中不断降低，最终达到稳定平衡状态。为了说明这一问题，下面以神经网络中的任一神经元 I 的能量变化为例进行说明其能量函数为：

$$E_i = -\frac{1}{2}\sum_{j=1,j\neq i}^{n} w_{ij}v_i v_j + b_i v_i$$

从 t 时刻至 t+1 时刻此神经元能量的变化量为：

$$\begin{aligned}\Delta E_i &= E_i(t+1) - E_i(t)\\ &= -\frac{1}{2}\sum_{j=1,j\neq i}^{n} w_{ij}v_i(t+1)v_j + b_i v_i(t+1) + \frac{1}{2}\sum_{j=1,j\neq i}^{n} w_{ij}v_i(t)v_j - b_i v_i(t)\\ &= -\frac{1}{2}[v_i(t+1) - v_i(t)]\left[\sum_{j=1,j\neq i}^{n} w_{ij}v_j + b_i\right]\end{aligned}$$

由二值函数的定义知：

- 当 $\sum_{j=1,j\neq i}^{n} w_{ij}v_j + b_i \leqslant 0$ 时，$v_i(t+1) = -1$，因此有 $\Delta E_i \leqslant 0$；
- 当 $\sum_{j=1,j\neq i}^{n} w_{ij}v_j + b_i \geqslant 0$ 时，$v_i(t+1) = 1$，同样有 $\Delta E_i \leqslant 0$。

所以，网络的能量变化量始终是非正的。因此按照同一规则进行状态更新的网络的能量变化量总是小于 0 的，也即：

$$\Delta E \leqslant 0\,,\quad E(t+1) \leqslant E(t)$$

所以在满足参数的条件下，Hopfield 网络总是朝着能量减小的方向演化，并且由于能量函数有下界而趋于稳定状态，此时的状态就是 Hopfield 网络的输出。

Hopfield 网络的能量函数含有全局极小点和局部极小点。将这些点作为记忆状态，就可以将 Hopfield 网络用于联想记忆；将能量函数作为目标代价函数，全局极小点看做最优解，就可以将 Hopfield 网络用于最优化计算。

13.1.2 连续 Hopfield 网络模型

连续 Hopfield 网络在拓扑结构上与离散 Hopfield 网络相同，只是神经元的传递函数采用连续函数。在连续 Hopfield 网络中，各神经元的输入、输出均为连续模拟量，各神经元采用并行工作模式，这种结构和生物神经系统中大量存在的神经反馈回路是一致的，并且非常适于利用模拟电子电路进行实现。1984 年，Hopfield 利用模拟电子电路实现了 Hopfield 网络，并将其应用于最优化问题的求解，成功地解决了 TSP 问题。

连续 Hopfield 神经网络结构模型用电路可以表示为图 13-3 所示的形式。

可以看到，网络中的每个神经元都可以看做是由运算放大器及其相关电路组成的，其中任意一个运算放大器 i（即神经元 i）都有两组输入：一组是恒定的外部输入，用 I_i 表示，

相当于运算放大器的电流输入；另一组是来自其他运算放大器的反馈连接，比如其中任意两个运算放大器 i 与 j 之间的连接权值，用 w_{ij} 表示。以 v_i 表示运算放大器的输出电压，u_i 表示运算放大器的输入电压，它们之间的关系为：

$$v_i = f(u_i)$$

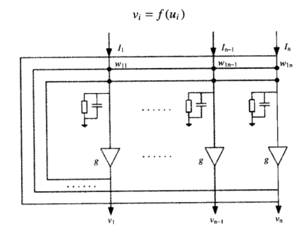


图 13-3　电路形式的连续 Hopfield 网络结构

其中 f 为神经元及运算放大器的传递函数，通常采用 sigmoid 函数，即：

$$f(u) = \frac{1}{(1+e^{u})}$$

连续 Hopfield 网络在时间和数值上都是连续的，各神经元处于同步工作模式下，利用电路理论中的基尔霍夫电流定律可以给出网络满足的方程如下：

$$C_i \frac{\mathrm{d}u_i}{\mathrm{d}t} = \sum_{j=1}^{N} w_{ij}(v_j - u_i) + I_i - \frac{u_i}{R_{i0}}$$

其中，$w_{ij} = \frac{1}{R_{ij}}$。对于稳定的连续 Hopfield 网络，权值矩阵同样需要满足对称性，即：

$$w_{ij} = w_{ji}\,,\quad w_{ii} = 0$$

连续 Hopfield 网络的李雅普诺夫能量函数定义如下：

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} v_i v_j + \sum_{i=1}^{N}\frac{1}{R_i}\int_0^{u_i} f^{-1}(v_i)dv_i - \sum_{i=1}^{N} I_i v_i$$

其中，$\frac{1}{R_i} = \frac{1}{R_{i0}} + \sum_{j=1}^{N} w_{ij}$，$f^{-1}$ 为传递函数的反函数。

可以证明，如果网络神经元的传递函数是 sigmoid 函数，并且网络连接权值矩阵为零对角对称阵，则网络的李雅普诺夫能量将随时间变化而单调下降或保持不变；并且，当且仅当输出电位随时间的变化不变时，网络能量不变。

另一方面，因为能量函数是有下界的，所以，连续 Hopfield 网络也是稳定的，其稳定点必定位于能量函数的下界，即极小值处。

连续 Hopfield 网络模型对生物神经元模型做了很多的简化，但仍然突出了生物神经系统的一些主要特性，包括以下几点。

（1）连续 Hopfield 网络的神经元传输特性具有 sigmoid 特性。

（2）在神经元之间以反馈的方式存在着大量的兴奋和抑制性互连，反映了真实生物神经系统的特性。

（3）既有代表产生动作电位的神经元，又有代表渐进方式工作的神经元，保留了动态和非线性这两个重要的计算特性。

连续 Hopfield 网络主要用于优化计算领域。在实际应用中，对于任何一个系统，其目标函数定义为能量函数，那么总可以通过连续 Hopfield 网络对其进行求解。

通过神经网络这一动态系统给出初始估计点，然后随着网络的运动传递找到能量函数的最小点，这样就可以利用连续 Hopfield 网络求解大量的最优化计算问题。Hopfield 网络的这一项能力是具有开拓性的。

13.1.3 联想记忆

联想记忆（Associative Memory）是人工神经网络理论的一个重要组成部分，将人工神经网络用于模拟生物神经系统的记忆功能，是模式识别、智能控制以及人工智能领域的一个重要课题。它主要利用神经网络的容错性，将不完整的、畸变的、污损的输入样本恢复为完整的原型。由于 Hopfield 网络能够模拟生物神经网络的记忆功能，因此常常被称为联想记忆网络。

所谓联想记忆，是指从一种事物联系到相关的另一种事物的能力。联想的基础是记忆，也就是将输入信息存储起来，再在某些需要的场合，按某种方式将相关信息取出。它类似计算机的存取信息的过程，计算机是按地址存取，而神经网络的信息存取则是基于内容的存取。记忆地址通过记忆内容的部分来识别，信息分布于生物记忆的内容中，而不是某个固定的地址。

Hopfield 网络通过神经元之间的权值得到事物间的联系，各神经元之间的权值共同表现为神经网络的联想记忆功能。这种分布式存储能够存储较多的模式，从而在一定程度上具有恢复不完整信息的功能。因而可以应用于信号复原、图像复原等领域。

联想记忆的工作过程通常分为两个阶段：一是记忆阶段，二是联想阶段。

1．记忆阶段

所谓的记忆阶段就是设置网络的权值，使其具有若干个稳定的平衡状态，这些稳定的平衡状态称为吸引子（Attractor），吸引子具有一定的吸引域（Basin of Attraction）。吸引子的吸引域就是能够稳定收敛到该吸引子的所有初始状态的基和，其大小用半径来描述。

从图形上来看，吸引子也就是网络能量函数的极小值点，如图 13-4 所示，记忆过程就是将记忆和存储的模式映射为网络吸引子的过程。

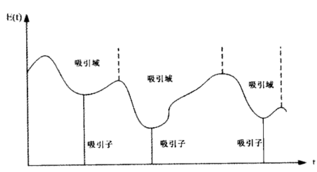


图 13-4　网络吸引子与吸引域示意

2. 联想阶段

联想过程就是给定输入模式，网络按照动力学的演化规则运行到稳定状态，收敛到吸引子，从而回忆起存储模式的过程。

对于一个具有联想记忆功能的网络，其吸引子的数量就代表了网络的记忆容量（Memory Capacity）或存储容量（Storage Capacity），也就是在一定的联想出错概率容限下，网络中存储互不干扰样本的最大输入。存储容量的大小与联想记忆的允许误差、网络结构、学习方式，以及设计参数有关。

另外，吸引域是衡量网络容错性的指标，吸引域越大，容错性就越好，或者说网络的联想能力就越强。

Hopfield 网络具有稳定的平衡状态，欲将其作为联想记忆网络之用时，需要设计和训练其权值，使吸引子存储记忆模式。常用的设计和学习算法包括外积法、投影学习法、伪逆法，以及特征结构法等。

设有 m 个样本存储向量，分别为 $\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_m$。其中：

$$\boldsymbol{x}_1=\{\boldsymbol{x}_{11},\boldsymbol{x}_{21},\ldots,\boldsymbol{x}_{m1}\}$$
$$\boldsymbol{x}_2=\{\boldsymbol{x}_{12},\boldsymbol{x}_{22},\ldots,\boldsymbol{x}_{m2}\}$$
$$\ldots$$
$$\boldsymbol{x}_m=\{\boldsymbol{x}_{1m},\boldsymbol{x}_{2m},\ldots,\boldsymbol{x}_{mm}\}$$

将这 m 个样本向量存储入 Hopfield 网络之中，则网络第 i，j 两个节点之间的权值为：

$$w_{ij}=\begin{cases}\sum \boldsymbol{x}_{ik}\cdot \boldsymbol{x}_{jk} & \text{当} i\neq j \text{时}\\ 0 & \text{当} i=j \text{时}\end{cases}$$

其中，k=1，2，…，m；i，j 分别是样本向量 $\boldsymbol{x}_k$ 的第 i，j 个分量 $\boldsymbol{x}_i$、$\boldsymbol{x}_j$ 的下标。

对联想存储器的检索过程如下。

① 初始化网络状态，给定向量 $\boldsymbol{X}=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_m\}$，将其设定为网络的初始状态。各个节点的初始状态分别为：

$$\boldsymbol{y}_j(0)=\boldsymbol{x}_j,\quad j=1,2,\ldots,m$$

② 按照迭代公式进行检索计算。公式为：

$$Y_j(t+1)=f[\sum w_{ij}y_j(0)-\theta_j]$$

其中 f 为传递函数。

③ 网络输出。迭代完毕，此时网络状态即为所需要的输出。

在此过程中，网络最终达到的状态就是利用给定向量进行联想记忆的结果。这个过程表明，即使是给定的输入向量并不完全或者不正确，也能够通过网络找到正确的结果。这等效于信号处理中的反卷积信号复原的效果。

13.1.4 Hopfield 网络结构

Hopfield 网络的结构如图 13-5 所示。

其中，$\boldsymbol{a}^1(k)=\text{satlins}(\boldsymbol{LW}^{1,1}\boldsymbol{a}^1(k-1)+\boldsymbol{b}^1)$，为 Hopfield 网络传递函数在 k 时刻的输出，当 k=0 时，$\boldsymbol{a}^1(0)=\boldsymbol{p}$。如前一节所述，输入 $\boldsymbol{p}$ 仅仅作为网络的初始条件。

Hopfield 网络的传递函数为饱和线性传递函数，函数名为 satlins，其传输曲线如图 13-6 所示。

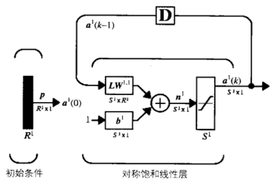


图 13-5 Hopfield 网络结构

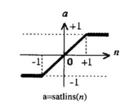


图 13-6 satlins 传递函数

当输入小于–1 之时，satlins 函数输出–1；当输入在–1 到+1 之间时，输出等于输入；当输入大于 1 时，函数输出为 1。

Hopfield 网络是一种多输入的非线性动态系统，以输入样本向量为初始条件，输出通过反馈回到输入端，在多次重复迭代之后使网络达到稳定状态。对应每一组输入向量即初始条件，网络都会收敛到对应的极小值点，这也是 Hopfield 网络联想记忆的原理。

13.2 Elman 反馈神经网络

Elman 反馈神经网络（以上简称为 Elman 网络）是一种带反馈的两层神经网络，反馈连接从第一层输出连接到输入端。此反馈连接使得 Elman 网络能够用来检测和生成时变模

式。一个双层 Elman 网络的结构如图 13-7 所示。

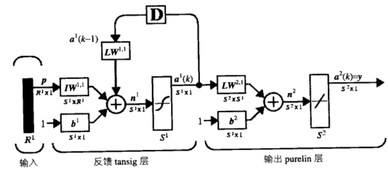


图 13-7　Elman 网络结构

其中：

$$\boldsymbol{a}^1(k)=\text{tansig}(\boldsymbol{LW}^{1,1}\boldsymbol{p}+\boldsymbol{LW}^{1,1}\boldsymbol{pa}^1(k-1)+\boldsymbol{b}^1),$$

$$\boldsymbol{a}^2(k)=\text{purelin}(\boldsymbol{LW}^{2,1}\boldsymbol{a}^1(k)+\boldsymbol{b}^2)$$

从图 13-7 中可以看到，Elman 网络的隐层也即反馈层的神经元采用 tansig 传递函数，而输出层采用线性传递函数。这种传递函数的组合使得 Elman 网络能够在有限的时间内以任意精度逼近任意函数（具有有限个不连续点）。唯一的要求是其隐层中必须包含足够多的神经元，隐层中神经元的数目越多，网络逼近复杂函数的能力就越强。

需要指出的是，Elman 网络与常规的两层 BP 网络的不同之处在于，Elman 网络的隐层是有反馈的。运行过程中，它保存上一步的运算结果，并反馈到输入端，应用于这一步的计算。因此，即使是两个具有相同权值和偏差的 Elman 网络，在一定的时间内给定相同的输入，由于两个网络具有不同的反馈状态，执行的最终结果可能也是不同的。

此外，Elman 网络也可以用做信息存储，因此 Elman 网络既支持学习空域模式，同时也支持学习时域模式。

13.3 反馈神经网络的 MATLAB 实现

本节我们介绍 MATLAB 神经网络工具箱中与反馈神经网络相关的主要函数。

13.3.1 设计 Hopfield 网络

MATLAB 工具箱中用于设计 Hopfield 网络的对应函数为 newhop。函数需要给定一组目标平衡点，以矩阵形式 $\boldsymbol{T}$ 表示，newhop 返回反馈网络的权值和偏差。生成的网络能够保证对于给定的目标期望向量包含稳定平衡点，但是也可能包含一些伪平衡点。设计中需要使得网络中的伪平衡点越少越好。

newhop 函数的调用格式为：

```
net = newhop(T)
```

其中，$\boldsymbol{T}$ 为代表给定的目标平衡点的向量。

网络生成之后，可以通过与一个或多个输入向量对其进行检验。如果输入向量与目标平衡点距离较近的话，将能够找到需要的目标平衡点。如同图 13-7 中显示的一样，输入向量是以批处理的模式整体输入的，网络将输出反馈到输入端，同时将输出向量与目标向量比较，从而决定将如何进行调整。

通过对输入向量进行批处理，可以较快地检测网络的性能。首先需要检验目标平衡点向量是否确实包含在网络中，其次可以试验其他输入向量，以此确定网络的目标平衡点的吸引域，以及网络是否存在伪平衡点。

下面举例说明 Hopfield 网络的设计过程。

【例 13-1】 Hopfield 网络设计实例。利用下面定义的目标样本向量，设计一个具有两个稳定平衡点的三维 Hopfield 网络。

解： ① 定义两个代表目标平衡点的三维目标向量。输入如下命令：

```
T = [-1 -1 1; 1 -1 1]'
```

输出结果为：

```
T =-1     1
   -1    -1
    1     1
```

② 调用 newhop 函数生成网络。输入命令：

```
net = newhop(T);
```

然后利用此平衡点目标向量检验设计的网络。输入命令：

```
Ai = T;
[Y,Pf,Af] = sim(net,2,[],Ai);
Y
```

对于网络没有输入的情形（如此处的 Hopfield 网络），需要将 sim 函数的第 2 个参数设为 2。输出保存在 Y 中，产生的结果如下：

```
Y =-1     1
   -1    -1
    1     1
```

可见，该网络确实能够在给定的设计点处得到稳定的平衡。

③ 下面重新给定网络初始输入条件，对网络进行仿真。输入命令：

```
Ai = {[-0.9; -0.8; 0.7]};
```

此输入向量与第一个设计点比较接近，因此可以预期网络将会收敛到第一个平衡点。输入命令如下查看结果：

```
[Y,Pf,Af] = sim(net,{1 5},{},Ai);
Y{1}
```

输出结果为：

```
ans =-1
   -1
    1
```

可以看到，网络确实如预期地收敛到了第一个稳定平衡点。我们希望网络对所有的输入都能够收敛到期望的稳定平衡点，但是由于 Hopfield 网络有可能存在伪平衡点，因此并不是总是可以得到期望的结果。这一点会在后面实例中进行介绍。

13.3.2 Elman 网络的创建与仿真

利用 MATLAB 神经网络工具箱函数 newelm 可以创建一个两层或多层的 Elman 网络。通常隐层中采用 tansig 传递函数，输出层采用 purelin 传递函数。

网络的默认 BP 训练函数为 trainbfg，也可以选择 trainlm，但是它的训练速度太快，在 Elman 网络中性能并不是很好。默认的 BP 学习规则为 learngdm，默认性能函数为 mse。

一旦网络创建完毕，其权值和偏差就通过 Nguyen-Widrow 方法实现初始化，对应函数为 initnw。下面举例说明。

【例 13-2】 Elman 网络设计实例。设计一个 Elman 网络并进行仿真。

解： ① 直接采用 newelm 命令生成网络。输入命令：

```
net = newelm([0 1],[5 1],{'tansig','logsig'});
```

此网络隐层中包含 5 个神经元，采用的传递函数为 tansig，输出层中只有一个神经元，采用的传递函数为 logsig，输入向量范围为[0, 1]。

采用随机数的方式生成一个输入向量，向量数值中只包含 0 和 1。输入命令：

```
P = round(rand(1,8))
```

输出结果为：

```
P = 1    1    0    1    0    0    0    1
```

将其转化为单元排列形式。输入命令：

```
Pseq = con2seq(P)
```

输出结果为：

```
Pseq =  [0]    [1]    [0]    [1]    [1]    [0]    [0]    [0]
```

② 利用 sim 函数对网络进行仿真，得到输出。输入命令：

```
Y = sim(net,Pseq)
```

输出结果为：

```
Y = [0.0589]    [0.3643]    [0.0946]    [0.4682]    [0.1861]    [0.7120]
    [0.2916]    [0.5146]
```

将结果 Y 转化为向量形式。输入命令：

```
z = seq2con(Y);
z{1,1}
```

输出结果为：

```
ans =0.0589    0.3643    0.0946    0.4682    0.1861    0.7120    0.2916
0.5146
```

输出结果表明网络还没有达到我们期望的性能。

13.3.3 训练 Elman 网络

可以利用 MATLAB 工具箱函数 train 或者 adapt 对 Elman 网络进行训练。

当应用 train 函数和 adapt 函数对网络进行训练时，每一次迭代过程都包括以下几个步骤。

① 将整个输入向量输入网络，计算得到其输出，同时与目标向量相比较，产生误差向量。

② 通过误差反向传播算法，得到误差函数对于各个权值和偏差的梯度。

③ 根据计算得到的梯度值，对权值与偏差进行修正。

函数 train 与 adapt 的不同之处在于，前者通常采用 BP 训练函数 traingdx 对网络进行训练，而后者则应用 learngdm 学习规则函数进行权值修正。

Elman 网络的可靠性比一些其他类型的网络要差，这是因为它在训练和调整过程中使用的都是误差梯度的近似值。

对于 Elman 网络应用于实际问题时，相比其他网络，会需要更多的隐层神经元，才能获得更好的精度。如果神经元数目较少的话，Elman 网络找到合适的权值与偏差的机会会更小，因为训练过程中采用的是近似梯度。因此，在创建 Elman 网络时给定足够数量的神经元是对输入进行正确划分的保证。

应用 train 函数对网络进行训练，需要提供输入向量与目标输出向量。利用有动量和可变学习速率的 BP 训练方法对网络进行训练。下面我们接上面的例子，对网络进行训练。

【例 13-3】 Elman 网络训练实例。设计 Elman 网络并利用输入与目标输出向量样本进行训练。

解： ① 给定训练需要的输入样本向量。输入命令如下：

```
P = round(rand(1,8))
```

输出结果为：

```
P = 1    0    0    0    1    1    1    0
```

然后定义目标输出向量。输入命令：

```
T = [0 (P(1:end-1)+P(2:end) == 2)]
```

输出结果为：

```
T = 0    0    0    0    0    1    1    0
```



此处，只有当输入向量中的元素值连续两个均为 1 时，输出才为 1，其余时候都为 0。

② 调用 newelm 函数生成一个隐层包含 5 个神经元的 Elman 网络。输入如下命令：

```
net = newelm([0 1],[5 1],{'tansig','logsig'});
```

应用 train 函数对网络进行训练，设定最大迭代次数为 100。训练结束后，对网络进行仿真，并查看输出结果。输入如下命令：

```
Pseq = con2seq(P);
Tseq = con2seq(T);
net.trainParam.epochs=100;
net = train(net,Pseq,Tseq);
Y = sim(net,Pseq);
z = seq2con(Y);
z{1,1},
diff1 = T - z{1,1},
```

输出结果为：

```
ans =0.1016    0.0585    0.0022    0.0117    0.0935    0.8468    0.8141
0.1269
diff1 =-0.1016   -0.0585   -0.0022   -0.0117   -0.0935    0.1532    0.1859
-0.1269
```

在弹出的 nntraintool 窗口中单击“Performance”按钮，可以查看训练过程中误差性能的变化，如图 13-8 所示。

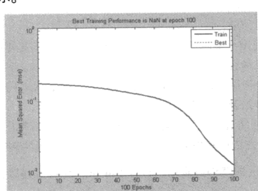


图 13-8 训练过程中网络误差性能的变化

从图 13-8 可以看到，网络输出比较好地收敛到期望的目标响应，最终的输出误差 diff1 较小，证明经过训练的 Elman 网络是能够解决该问题的。

13.4 反馈神经网络应用实例

13.4.1 二神经元 Hopfield 网络设计

【例 13-4】 二神经元的 Hopfield 网络设计实例。利用给定的样本向量设计一个二神

经元的 Hopfield 网络，该网络具有两个稳定平衡点，然后对设计的网络进行仿真。

解：① 定义训练需要的输入样本向量。输入如下命令：

```
T = [+1 -1; ... -1 +1];
```

作图显示输入样本向量。输入命令：

```
plot(T(1,:),T(2,:),'r*')
axis([-1.1 1.1 -1.1 1.1])
title('Hopfield Network State Space')
xlabel('a(1)'); ylabel('a(2)');
```

绘出的图形如图 13-9 所示。

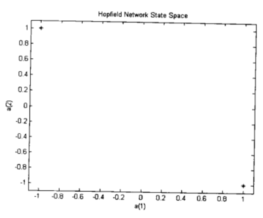


图 13-9 Hopfield 网络目标稳定平衡点

可以看到，网络两个预定的稳定平衡点位于状态空间图的左上角和右下角。

② 利用 newhop 函数创建 Hopfield 网络。输入命令：

```
net = newhop(T);
```

可以用目标向量对网络进行测试，如果生成的网络没有问题，应该会收敛到预定的平衡点位置。输入命令：

```
[Y,Pf,Af] = sim(net,2,[],T);
Y
```

输出结果为：

```
Y = 1   -1
   -1    1
```

确实如我们所料，Hopfield 网络收敛到了正确的平衡点。

③ 下面随机选取初始点，利用 sim 函数对网络进行仿真，设定网络运行步数为 20 步，然后查看网络状态是否收敛到预定平衡点。输入命令：

```
a = {rands(2,1)};
[y,Pf,Af] = sim(net,{1 20},{},a);
```

对 Hopfield 网络的运行过程绘图显示。输入命令:

```
record = [cell2mat(a) cell2mat(y)];
start = cell2mat(a);
hold on
plot(start(1,1),start(2,1),'bx',record(1,:),record(2,:))
```

绘出的图形如图 13-10 所示，可以看到，网络确实收敛到了期望的平衡点。

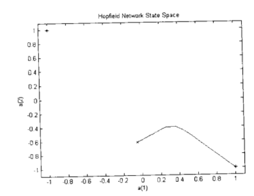


图 13-10　Hopfield 网络运行情况

④ 选择 25 个新的初始点，再次仿真，并查看网络运行情况。输入命令:

```
color = 'rgbmy';
for i=1:25
   a = {rands(2,1)};                              %设置随机初始点
   [y,Pf,Af] = sim(net,{1 20},{},a);              %网络仿真
   record=[cell2mat(a) cell2mat(y)];              %记录演化轨迹
   start=cell2mat(a);
   plot(start(1,1),start(2,1),'kx', ...
   record(1,:),record(2,:),color(rem(i,5)+1))     %绘制演化轨迹图
end
```

绘出的图形如图 13-11 所示。

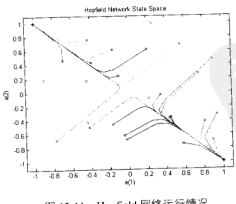


图 13-11　Hopfield 网络运行情况

可以看到距离左上角近的初始状态，最终都收敛到左上角的平衡点，而距离右下角近的初始状态，最终都收敛到了右下角的平衡点。这说明 Hopfield 网络确实有联想记忆的能力。

13.4.2 Hopfield 网络中的伪平衡点

如前面所述，Hopfield 网络中可能会存在伪平衡点，导致网络收敛到非期望状态，下面举例说明。

【例 13-5】 考虑【例 13-4】中二神经元的 Hopfield 网络，用特定向量仿真，考察其平衡点情况。

解： ① 定义目标向量。输入命令：

```
T = [1 -1; -1 1]'
```

输出结果为：

```
T = 1    -1
   -1     1
```

绘图显示网络状态空间的平衡点。输入如下命令：

```
plot(T(:,1),T(:,2),'*');axis([-1.2, 1.2, -1.2, 1.2]);
xlabel('a(1)'); ylabel('a(2)')
```

绘出图形如图 13-12 所示。

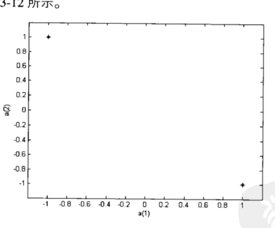


图 13-12 Hopfield 网络目标稳定平衡点

② 调用 newhop 函数生成 Hopfield 网络。输入命令：

```
net = newhop(T);
```

然后查看生成网络的权值和偏差。输入命令：

```
W=net.LW{1,1},
B=net.b{1,1},
```

输出结果为：

```
W = 0.6925   -0.4694
   -0.4694    0.6925
B =  0
     0
```

③ 利用目标向量 T 对网络进行测试，并查看网络的目标输出。输入命令：

```
Ai = T;
[Y,Pf,Af] = sim(net,2,[],Ai);
Y
```

输出结果为：

```
Y =  1    -1
    -1     1
```

如我们所预期，网络确实收敛到了期望的平衡点。

④ 现在随机选取初始点，利用 sim 函数对网络进行仿真，看它是否能够稳定到期望的平衡点。输入命令：

```
a = {rands(2,1)};
[y,Pf,Af] = sim(net,{1 50},{},a);
```

接下来绘制网络活动过程。输入命令：

```
record = [cell2mat(a) cell2mat(y)];
start = cell2mat(a);
hold on
plot(start(1,1),start(2,1),'ro',record(1,:),record(2,:))
```

绘出的图形如图 13-13 所示。

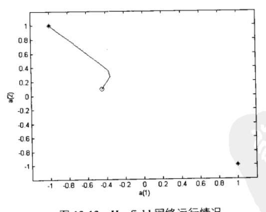


图 13-13　Hopfield 网络运行情况

可以看到，运行之后，随机初始点最终稳定在了左上角的平衡点。在这一次测试中，网络运行收敛的结果是符合期望的。我们可以加入更多的测试点观察网络运行情况。

增加 5 个初始输入点，对网络进行测试。注意此处选取的 5 个初始点恰好位于左下-

右上对角线上，与两个平衡点距离相等。输入命令：

```
plot(0,0,'ko');
P = [-1.0 -0.5 0.0 +0.5 +1.0;
     -1.0 -0.5 0.0 +0.5 +1.0];
color = 'rgbmy';
for i=1:5
   a = {P(:,i)};                                %设置初始点
   [y,Pf,Af] = sim(net,{1 50},{},a);            %网络仿真
   record=[cell2mat(a) cell2mat(y)];            %记录演化轨迹
   start = cell2mat(a);
   plot(start(1,1),start(2,1),'kx', ...
   record(1,:),record(2,:),color(rem(i,5)+1))   %绘制演化轨迹图
   drawnow
end
```

输出的图形如图 13-14 所示。

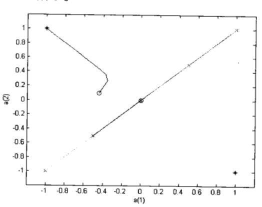


图 13-14 Hopfield 网络的伪平衡点

图 13-14 中，新增加的 5 个输入点都位于左下-右上对角线上。可以看到，随着网络运行，5 个初始点最终都收敛到了状态空间的中心点，然而这一点并不是我们期望的网络平衡点，因此也说明了设计的 Hopfield 网络中是包含了伪平衡点的，这是 Hopfield 网络的一个缺点。

13.4.3 三神经元 Hopfield 网络设计

【例 13-6】 三神经元的 Hopfield 网络设计实例。利用给定的样本向量设计一个三神经元的 Hopfield 网络，该网络具有两个稳定平衡点，然后对设计的网络进行仿真。

解： ① 定义两个三维的目标向量，对应网络的两个期望平衡点。输入命令：

```
T = [+1 +1; ... -1 +1; ... -1 -1];
```

对这两个目标向量在网络状态空间中的位置进行绘图显示。输入如下命令：

```
axis([-1 1 -1 1 -1 1])
```

```
set(gca,'box','on'); axis manual;  hold on;
plot3(T(1,:),T(2,:),T(3,:),'r*')
title('Hopfield Network State Space')
xlabel('a(1)'); ylabel('a(2)'); zlabel('a(3)');
view([37.5 30]);
```

绘出的图形如图 13-15 所示。

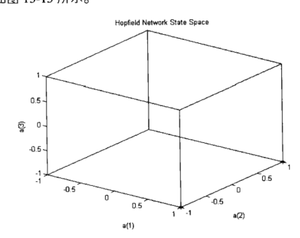


图 13-15　生成的 Hopfield 网络状态空间

可以看到，网络两个平衡点位于状态空间立方体下部的两个顶点处。

② 利用函数 newhop 创建以上述两点为平衡点的 Hopfield 网络。输入命令：

```
net = newhop(T);
```

随机定义一个初始点，对生成的 Hopfield 网络进行仿真，预期网络经过运行将会达到预期的稳定平衡点。输入命令：

```
a = {rands(3,1)};
[y,Pf,Af] = sim(net,{1 10},{},a);
```

对网络的运行情况进行绘图显示。输入命令：

```
record = [cell2mat(a) cell2mat(y)];
start = cell2mat(a);
hold on
plot3(start(1,1),start(2,1),start(3,1),'bx', ...
   record(1,:),record(2,:),record(3,:))
```

绘出的图形如图 13-16 所示。可以看到，网络经过运行收敛到了其中一个稳定平衡点。

③ 随机生成 25 个初始状态点，进行仿真，并绘图观察网络运行过程。输入命令：

```
color = 'rgbmy';
for i=1:25
   a = {rands(3,1)};                    %设置随机三维初始点
   [y,Pf,Af] = sim(net,{1 10},{},a);    %网络仿真
   record=[cell2mat(a) cell2mat(y)];    %记录演化轨迹
```

```
    start=cell2mat(a);
    plot3(start(1,1),start(2,1),start(3,1),'kx', ...
        record(1,:),record(2,:),record(3,:),color(rem(i,5)+1)) %绘制演化轨迹
end
```

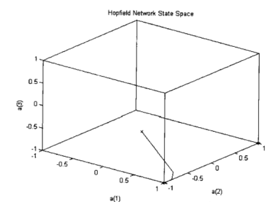


图 13-16 网络仿真运行情况

绘出的图形如图 13-17 所示。

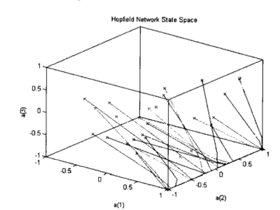


图 13-17 给定随机初始点的网络仿真运行情况

可以看到，在所有的随机初始状态下，网络最终都收敛到了期望的两个稳定平衡点。

④ 现在选择一系列特殊的初始状态点，这些初始状态点恰好位于与两个初始平衡点距离相等的对角边线上，如图 13-18 所示，对网络进行仿真，观察运行情况。输入命令：

```
P = [ 1.0  -1.0  -0.5  1.00  1.00  0.0; ...
      0.0   0.0   0.0  0.00  0.00 -0.0; ...
     -1.0   1.0   0.5 -1.01 -1.00  0.0];
cla
plot3(T(1,:),T(2,:),T(3,:),'r*')
color = 'rgbmy';
for i=1:6
```

```
    a = {P(:,i)};                                   %设置三维初始点
    [y,Pf,Af] = sim(net,{1 10},{},a);              %网络仿真
    record=[cell2mat(a) cell2mat(y)];               %记录演化轨迹
    start=cell2mat(a);
    plot3(start(1,1),start(2,1),start(3,1),'kx', ...
       record(1,:),record(2,:),record(3,:),color(rem(i,5)+1))  %绘制演化轨迹
end
```

绘出的图形如图 13-18 所示。

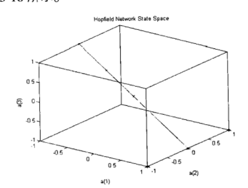


图 13-18　给定特殊初始状态点的网络仿真运行情况

可以看到，在这些初始状态下，Hopfield 网络并没有收敛到期望的两个稳定平衡点，而是收敛到边线中点所对应的网络状态上。这表明设计的三维 Hopfield 网络产生了不期望的稳定平衡点，也即伪平衡点。

13.4.4 利用 Elman 网络进行振幅检测

Elman 网络可以用来识别或产生空间和时间信号。其中，振幅检测就是应用 Elman 网络对时域信号进行检测和分类的例子。这一节我们利用实例对其进行说明。

振幅检测需要在神经网络输入端加入一个具有特定振幅的波形，然后由网络提取波形的振幅特征并输出。这是一个比较简单的应用实例，但是可以用来较清晰地说明 Elman 网络的设计方法和功能。

【例 13-7】　Elman 网络振幅检测应用实例。设计一个 Elman 网络，对输入的振幅变化的时变信号进行振幅检测与模式分类。

解： ① 首先定义两个正弦信号，其中一个振幅为 1，另一个振幅为 2。输入命令：

```
p1 = sin(1:20);
p2 = sin(1:20)*2;
```

然后定义期望输出为其振幅。输入命令：

```
t1 = ones(1,20);
t2 = ones(1,20)*2;
```

将输入、输出分别合并成一个序列，并重复一次，利用这个较长的波形作为 Elman 网

络的训练样本向量。输入命令：

```
p = [p1 p2 p1 p2];
t = [t1 t2 t1 t2];
```

将输入、输出由矩阵形式转换为序列形式。输入命令：

```
Pseq = con2seq(p);
Tseq = con2seq(t);
```

② 创建 Elman 网络，对于每一步输入的波形信号，输出其振幅。此网络为单输入、单输出结构。反馈层可以有任意个神经元，这要根据问题的复杂度而定。如果期望网络性能较好的话，就需要在反馈层中加入更多的神经元。

利用 newelm 函数创建网络，由于此问题相对来说比较简单，因此在反馈层中设置 10 个神经元，同时定义网络的输入、输出变动范围为[-2, 2]，训练函数为可变学习速率训练算法，对应函数为 traingdx，输入、输出层传递函数分别为 tansig 和 purelin。输入命令：

```
net = newelm([-2 2],[10 1],{'tansig','purelin'},'traingdx');
```

设定最大训练迭代次数为 1000，期望目标误差为 0.01。输入命令：

```
net.trainParam.epochs = 1000;
net.trainParam.goal = 0.01;
```

③ 利用 train 函数对网络进行训练。输入命令：

```
net = train(net,Pseq,Tseq);
```

除了单击 nntraintool 窗口上的“Performace”按钮绘制误差曲线以外，也可以利用命令方式绘制误差曲线。输入命令：

```
semilogy(tr.epoch,tr.perf)
title('Mean Squared Error of Elman Network')
xlabel('Epoch')
ylabel('Mean Squared Error')
```

绘制的误差曲线如图 13-19 所示。

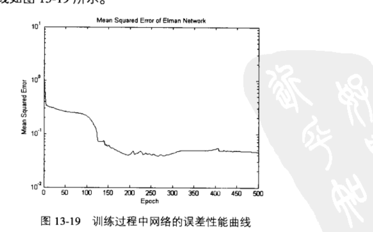


图 13-19 训练过程中网络的误差性能曲线

可以看到，经过训练网络的均方误差水平降到了 1e–2 量级。如果增加训练时间，网络性能还可以进一步改善。

④ 下面对网络进行仿真测试，以原始输入样本作为测试向量，调用 sim 函数完成仿真并绘图。输入如下命令：

```
a = sim(net,Pseq);
```

然后对其进行绘图显示。输入命令：

```
time = 1:length(p);
plot(time,t,'--',time,cat(2,a{:}))
title('Testing Amplitute Detection')
xlabel('Time Step')
ylabel('Target - -  Output ---')
```

输出图形如图 13-20 所示。

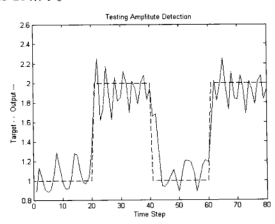


图 13-20 网络仿真输出结果与期望响应

其中，期望响应用虚线表示，实际输出用实线表示。可以看到，网络对输入信号振幅的检测是完成得比较好的，如果延长训练时间的话，网络的性能可以进一步提高；另一方面，增加隐层中神经元的数目也可以改善网络的性能。

⑤ 下面利用新的振幅分别为 1.6 与 1.2 的正弦波形输入对网络进行仿真，查看网络在遇到训练中不曾遇到过的输入时能否进行正确的振幅检测。输入命令：

```
p3 = sin(1:20)*1.6;
t3 = ones(1,20)*1.6;
p4 = sin(1:20)*1.2;
t4 = ones(1,20)*1.2;
pg = [p3 p4 p3 p4];
tg = [t3 t4 t3 t4];
pgseq = con2seq(pg);
```

利用序列 pgseq 和目标序列对网络进行仿真。输入命令：

```
a = sim(net,pgseq);
```

对结果进行绘图显示。输入命令：

```
time = 1:length(pg);
plot(time,tg,'--',time,cat(2,a{:}))
title('Testing Generalization')
xlabel('Time Step')
ylabel('Target - - Output ---')
```

绘出图形如图 13-21 所示。

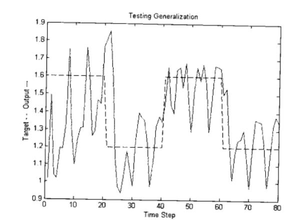


图 13-21　新输入样本下的网络仿真输出结果与期望响应

图中期望响应用虚线表示，实际输出用实线表示。可以看到，网络对输入信号进行振幅检测的效果并不是很好，网络输出摆动幅度很大，并且不能与期望响应很好的契合。这是因为训练样本中不包含新的输入样本，也就是说，对于新的输入样本，网络是未经过训练的，对于这种情况，利用更多不同振幅的信号对网络进行训练可以改善网络性能。

13.5 小结

MATLAB 神经网络工具箱中，包括两种反馈网络，即 Hopfield 网络和 Elman 网络。本章对这两种网络模型结构、训练方法以及设计仿真分别进行了介绍，然后通过实例介绍反馈网络的应用。

Part 3

神经网络综合实战篇

第 14 章 神经网络优化

前面各章介绍了几种基本的神经网络类型，利用它们还可以衍生一些优化的网络类型。例如，对于 BP 网络，为了改善收敛速度和改进其局部极小点，人们提出了不同于基本训练算法的多种学习算法；为了寻求更好的模式识别性能，提出了基于有限样本二次问题最优解的支持向量机算法；同时还提出了针对神经网络全值的遗传算法优化等。这些方法的发展，进一步丰富了神经网络的功能，同时也使得其训练过程速度更快，达到最优性能的概率更高。这些优化措施和优化类型也是提高神经网络性能的方向。

BP 网络的训练算法优化在前面章节中已经有过介绍，本章将对支持向量机、Boltzman 机与模拟退火算法，以及神经网络的遗传算法优化等相关理论进行介绍。

14.1 支持向量机

支持向量机是为了改善前向网络的几个问题而发展起来的。其中主要包括以下几个问题。

（1）BP 训练算法基于梯度算法，因此优化过程可能陷入局部极小值。

（2）学习目标是分类误差对所有样本点都达到最小，在支持向量机理论中，这属于经验风险，只能够保证分类误差对有限个样本是最小的，还有改善的余地。

（3）神经网络的结构设计依赖于设计者的先验知识（例如隐层中神经元的数目），缺乏科学的设计方法。

（4）无法控制是否收敛以及收敛的速度。

支持向量机是由 Vapnik 等人于 1959 年在统计学习理论基础上提出来的，是模式识别的一种新方法。它主要是根据有限的样本信息在模型的复杂性和学习能力之间寻求一个最佳的折中，在形式上类似于多层前向网络，但是能够克服一些多层前向网络的固有缺陷。它主要具有下列几个优点。

（1）专门针对有限样本情况而设计，目标是得到现有信息下的最优解而不是样本趋于无穷大时的最优解。

（2）算法最终转化为一个二次型最优求解问题，理论上可以得到全局最优解。

（3）算法将实际问题通过非线性变换转换到高维特征空间，在高维空间中构造线性判别函数来实现原空间的非线性判别函数，这一特殊性质保证支持向量机具有好的泛化能力，并且巧妙地解决了维数问题，使得样本复杂度与维度无关。

基于统计学习的支持向量机方法能够从理论上实现对不同类别的最优划分，具有很好的泛化性能。

14.1.1 统计学习理论

传统统计学研究的是无限样本情况下网络的极限特性，而 Vapnik 等人提出的基于有限样本的统计学习理论则专门研究小样本情况下网络的学习规律的理论，从更本质上研究其学习问题，从而克服前向网络的固有缺陷。

机器学习的目标是根据给定的训练样本估计系统输入与输出之间依赖关系，它能够对未知样本的输出做出尽可能准确的预测。因此，可以定义下面的风险函数，对学习效果进行评估。

对 n 个相互独立并服从同一分布的观测样本，假定系统在给定输入为 x 的情况下输出为 y，且变量 y 与 x 之间存在依赖关系，即遵循未知的联合概率 $F(x,y)$，则学习机器的期望风险定义为：

$$R(w)=\int L(y,f(x,w))\mathrm{d}F(x,y)$$

其中 $\{f(x,w)\}$ 称为预测函数集，w 为函数的广义参数，$L(y,f(x,w))$ 则称为损失函数。

具体的学习目标设定为使得风险函数最小，在实际应用中，由于样本数目有限，因此通常用 $R_{emp}(w)=\frac{1}{n}\sum_{i=1}^{n}L(y_i,f(x_i,w))$ 来求期望风险的最小值，也即经验风险最小化原则。经验风险最小化原则是目前绝大多数模式识别方法的基础。

Vapnik 等人通过研究得出结论，对指示函数（取值只有 0 和 1）中的所有函数，经验风险和实际风险之间以一定概率 $1-\eta$ 满足如下关系：

$$R(w)=R_{emp}(w)+\sqrt{\frac{h(\ln(2l/h)-\ln(\eta/4))}{l}}$$

其中，h 是函数集的 VC 维，VC 维定义为学习机器能够正确划分的任意二值标识的最大点集样本数目。它是衡量学习机器复杂程度的指标，VC 维越大表示此学习机器越复杂。l 是样本数。

上面等式右边前一部分称为经验风险，也称做训练误差；后一部分称为置信范围，置信范围不仅受 VC 维数的影响，同时也是置信水平 $1-\eta$ 的函数，可以记为 $\Phi(h/l)$。通常的训练是使得经验风险最小，但是由于样本可能不充分，造成置信范围非常大，导致机器的适应范围很窄，也就是泛化能力很差。

为解决上述问题，就需在保证分类精度的同时，降低学习机器的 VC 维，从而使得学习机器在整个样本集上的期望风险得到控制，实现结构风险最小化（Structural Risk Minimization，SRM）。这正是支持向量机实际所采用的方法。

14.1.2 支持向量机（SVM）理论

支持向量机理论最初来源于对数据分类的处理。对于线性可分的二值分类，通常是构造一个超平面并移动它，直到找到一个合适的分界线为止。这种激励不能保证分割平面位

于两个类别的中心，也就是说，无法最优化决策分界线。

支持向量机则很好地解决了这一问题，能够寻找一个最优分类面，使得该超平面在保证分类正确性的同时，平面两边的空白区域最大化。支持向量机分为线性和非线性两类。下面分别介绍。

1．线性支持向量机

对于线性可分问题，如图 14-1 中所示的上下两组样本点，通过平行直线 $H1$，H，$H2$ 都可以成功地分开（其他斜率的平行线也与之类似）。$H1$ 和 $H2$ 这样的两类直线之间的距离称为分类间隔，相比于 $H1$ 和 $H2$，H 的分类是更优的。

所谓最优分类线，是使两类的分类间隔都最大化的分类线。能够把两类样本点正确无误地分开，是保证经验风险最小化，而能够使分类间隔最小化，则是使得置信范围最小，这就是结构风险最小化的体现。在高维空间中，最优分类线就成为了最优分类面。

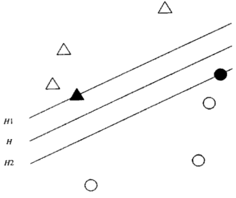


图 14-1　最优分界线示意图

设训练样本为 x_i，i=1，2，…，l，对应的期望输出为 $y_i=\{+1,-1\}$，其中+1 和−1 代表两类的类别标识。假定分类面方程为 $wx+b=0$。为了使得分类面为所有样本正确分类且具备分类间隔，就要满足下列约束条件：

$$y_i(x_i w+b)-1\geqslant 0$$

可以计算出分类间隔为：

$$\min_{\{x_i|y_i=1\}}\frac{x_i w+b}{\|w\|}-\max_{\{x_i|y_i=-1\}}\frac{x_i w+b}{\|w\|}=\frac{2}{\|w\|}$$

训练的目标就是使得分类间隔 $\frac{2}{\|w\|}$ 最大化，也即 $\|w\|^2$ 最小化。于是将最优分类问题转化为下列约束优化问题，求下式的最小值：

$$\Phi(w)=\frac{1}{2}\|w\|^2$$

而约束条件为：

$$y_i(x_i w+b)-1\geqslant 0$$

采用拉格朗日乘子法，将问题转化为：

$$L=\frac{1}{2}\|w\|^2-\sum_{i=1}^{l}\alpha_i y_i(x_i w+b)+\sum_{i=1}^{l}\alpha_i$$

因此，目标就转化为求 L 的最小值。因此，分别对 w 和 b 求偏微分，令其等于 0，则转化为一个简单的问题，对 α_i 求下面函数的最大值：

$$W(\alpha)=\sum_{i=1}^{l}\alpha_i-\frac{1}{2}\sum_{i=1}^{l}\alpha_i\alpha_j y_i y_j(x_i x_j)$$

其约束条件为 L 关于 w 和 b 的梯度均为 0，同时满足 $\alpha_i\geqslant 0$。

如果 α_i^* 为最优解，则：

$$w^*=\alpha_i^* y_i x_i$$

即为最优分类面的权系数向量，此权向量是训练样本的线性组合。

上述问题是一个不等式约束下求二次极值的问题，可以证明，其最优解必须满足：

$$\alpha_i^*\{[x_i w+b]y_i-1\}=0$$

对于多数样本，α_i^* 取 0，而对取值不为 0 的 α_i^*，则必须使得 $y_i(x_i w+b)-1\geqslant 0$ 等号成立，这些样本就称为支持向量（Support Vectors）。也就是图 14-1 中标记为黑色的样本点。

2．非线性支持向量机

解决感知器无能为力的异或问题，一种方法是使用多层前向网络，另一种方法则是将输入向量映射到一个高维的特征向量空间，并且在该特征空间中构造最优分类面，这就是支持向量机方法，它能够避免在多层前向网络中无法克服的一些缺陷。如果选用的映射函数合适的话，在大多数输入空间线性不可分的问题是可以在高维的特征空间中进行线性划分的。

在低维空间向高维空间的转化过程中，由于空间维数的增长，导致多数情况下难以直接在特征空间直接计算最佳分类平面，而支持向量机通过定义核函数，将这一问题巧妙的转化到输入空间进行计算。

在引入核函数的情况下，支持向量机的形式如图 14-2 所示。

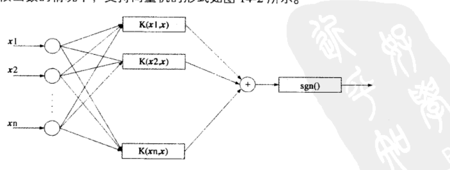


图 14-2 核函数支持向量机原理

其中，$K(\boldsymbol{x}_i,\boldsymbol{x}_j)$ 即为核函数，核函数定义了两个输入向量之间的内积。常用的核函数包括多项式核函数、高斯核函数、Sigmoid 核函数等。对应输出函数可以表示为：

$$f(\boldsymbol{x})=\mathrm{sgn}\left[\sum_{i=1}^{l} y_i\alpha_i^* K(\boldsymbol{x}_i,\boldsymbol{x})+b^*\right]$$

可以看出，支持向量机求得的决策函数形式上类似于一个神经网络，其输出是若干中间层节点的线性组合，而每一个中间层节点对应于输入样本与一个支持向量的内积。

当采用径向基核函数时，支持向量机就是一种径向基函数分类器，它与前面的传统径向基方法的区别在于使用了支持向量方法，径向基函数的中心位置以及中心数目、网络权值都是由训练过程中自动确定的。

14.1.3 支持向量机实例

MATLAB 神经网络工具箱中并没有支持向量机的函数和实例，运行本节中的实例需要安装本书附带光盘中的支持向量机 SVM 工具箱，此工具箱为第三方开放工具箱，也可以从 http://www.support-vector.net/software.html 下载。

在 MATLAB 中安装工具箱的过程非常简单，执行步骤如下。

(1) 解压缩 SVM-KM.zip 到期望的文件夹，最好选择 MATLAB 安装目录下的 toolbox 文件夹。

(2) 在命令窗口中选择“File”菜单下的“Set Path”选项，弹出如图 14-3 所示的窗口。

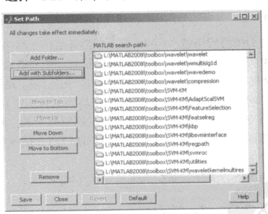


图 14-3 “Set Path”窗口

(3) 单击“Add with Subfolders”按钮，在其中选中解压缩后的“SVM-KM”文件夹，将其添加到 MATLAB 配置目录中。

(4) 单击“Save 按钮保存配置，然后单击“Close”按钮，即完成此工具箱的安装。

安装了工具箱之后，就可以进入下面的实例操作。

【例 14-1】 支持向量机分类实例。应用支持向量机对 100 个二维高斯分布的随机样

本点进行分类。

解： ① 清空数据空间。在 MATLAB 命令空间中输入命令：

```
close all
clc
```

接下来创建数据样本，输入命令：

```
n = 100;
sigma=0.3;
[Xapp,yapp,xtest,ytest]=datasets('gaussian',n,0,sigma);
Xapp=single(Xapp);
[Xapp]=normalizemeanstd(Xapp);
```

上述命令中调用了 SVM-KM 工具箱中的函数，产生 100 个高斯分布的随机样本数据点集，这些数据点位于 Xapp 中。输入下面命令对样本点集进行绘图显示：

```
plot(Xapp(:,1),Xapp(:,2),'.'),axis([-3.5,3.5,-3.5,3.5])
```

绘出的图形如图 14-4 所示。

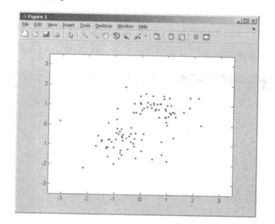


图 14-4　输入样本点集

② 下面调用 svmclass 命令通过支持向量机对输入样本点集进行划分。

```
lambda = 1e-7;
C = 10;
kernel='gaussian';
kerneloption=1;
[xsup,w,w0,pos,tps,alpha]=svmclass(Xapp,yapp,C,lambda,kernel,kernelopti
on,1);
ypredapp = svmval(Xapp,xsup,w,w0,kernel,kerneloption,1);
```

设置 SVM 核函数为 gausssian 函数，拉格朗日乘子边界为 10，lambda 是 QP 算法中用到的条件参数，输出支持向量即为 xsup，w 与 w0 分别是权值和偏差值。对于 MATLAB 中的命令 svmval 的调用格式，读者可以通过 type 命令查看源文件。

③ 创建一个二维网格，同时对判决函数进行验证。输入命令：

```
[xtest1 xtest2]  = meshgrid([-1:.05:1]*3.5,[-1:0.05:1]*3);
nn = length(xtest1);
Xtest = [reshape(xtest1 ,nn*nn,1) reshape(xtest2 ,nn*nn,1)];
ypred=svmval(Xtest,xsup,w,w0,kernel,
kerneloption,[ones(length(Xtest),1)]);
ypred = reshape(ypred,nn,nn);
```

④ 绘图显示分类结果。输入如下命令:

```
figure(1);
clf;
%contourf(xtest1,xtest2,ypred,50);shading flat;
hold on
[cc,hh]=contour(xtest1,xtest2,ypred,[-1  0 1],'k');
clabel(cc,hh);
set(hh,'LineWidth',1);
h1=plot(Xapp(yapp==1,1),Xapp(yapp==1,2),'+r');
set(h1,'LineWidth',1);
h2=plot(Xapp(yapp==-1,1),Xapp(yapp==-1,2),'db');
set(h2,'LineWidth',1);
h3=plot(xsup(:,1),xsup(:,2),'ok');
set(h3,'LineWidth',1);axis([-3.5 3.5 -3 3]);
```

绘出的图形如图 14-5 所示。

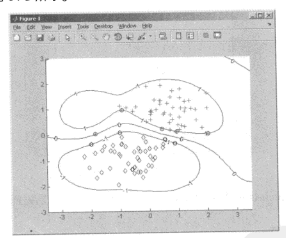


图 14-5 SVM 分类结果

从图中可以清楚地看到 SVM 方法确定的最优的决策分界线，以及+1，−1 两条等值线的形状。

下面介绍一个应用支持向量机实现一维回归的实例。

【例 14-2】 支持向量机回归分析实例。应用支持向量机对一维指数正弦函数样本点进行回归拟合。

解: ① 清空数据空间，定义一维样本数据，并对其绘图显示。在 MATLAB 命令空间中

输入命令：

```
clear all
tic
n=50;
x=linspace(0,3,n)'; xx=linspace(0,3,n)';
xi=x;
yi=cos(exp(x)); y=cos(exp(xx));
figure(1)
hold on;
plot(xx,y,'g',xx,y,'b+');
```

绘出图形如图 14-6 所示。

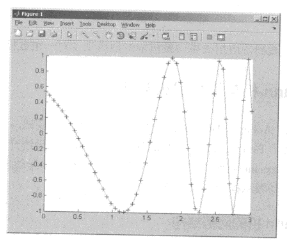


图 14-6 一维输入数据样本

② 设置参数，并调用 svmreg 函数进行回归。输入如下命令：

```
C = 0.0001; lambda = 0.000001;
epsilon = .2;
kerneloption = 0.01;
kernel='gaussian';
verbose=1;
[xsup,ysup,w,w0]=svmreg(xi,yi,C,epsilon,kernel,
kerneloption,lambda,verbose);
```

③ 调用 svmval 函数对数据进行绘图显示。输入命令：

```
rx = svmval(xx,xsup,w,w0,kernel,kerneloption);
h = plot(xx,rx,'b'); set(h,'LineWidth',2);
h = plot(xsup,ysup,'or'); set(h,'LineWidth',2);
plot(xx,rx+epsilon,'b--');
plot(xx,rx-epsilon,'b--'); hold off;
title('Support Vector Machine Regression');
xlabel('x');ylabel('y'); hold off
```

绘出图形如图 14-7 所示。



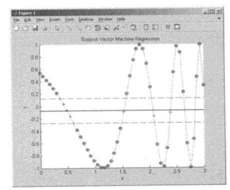


图 14-7 SVM 回归结果

从图 14-7 中可以看到，应用支持向量机很好地实现了对一维数据样本的回归拟和。

14.2 Boltzmann 机与模拟退火算法

20 世纪 80 年代，Ackley、Hinton 等人以模拟退火算法为基础，在 Hopfield 模型中引入了随机机制，提出了 Boltzmann 机。Boltzmann 机是第一个受统计力学启发的多层学习机，其命名来源于 Boltzmann 在统计力学中的早期工作与网络本身动态分布行为的相似性。Boltzman 机的运行机制服从模拟退火算法。

14.2.1 Boltzmann 机的网络结构

Boltzmann 机结合了 BP 网络和 Hopfield 网络在网络结构、学习算法和动态运行机制上的优点，具有多层的网络结构、高效的学习算法，以及依概率方式工作的动态运行机制。

Boltzmann 机由输入部、输出部和中间部构成。输入部和输出部神经元通常称做显见神经元，是网络与外部交换的媒介；中间部神经元称为隐见神经元。但是 Boltzmann 机没有明显的网络层次。此外，Boltzmann 机的神经元是互联的，网络状态按照概率分布进行变化。其结构如图 14-8 所示。

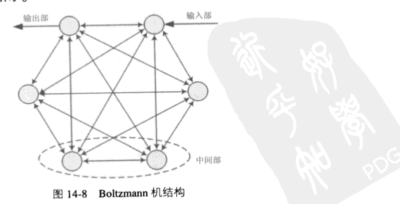


图 14-8 Boltzmann 机结构

同 Hopfield 网络一样，Boltzmann 网络中的每一对神经元之间的信息传递都是双向对称的，即 $w_{ij}=w_{ji}$，并且自身不存在反馈，即 $w_{ii}=0$。在网络学习期间，显见神经元被外部环境所约束，因而处于某一特定的状态之下，而中间部隐见神经元则不受外部环境的约束。

Boltzmann 机中的每一个神经元都具有随机的兴奋和抑制状态，其概率取决于神经元的输入。神经元结构如图 14-9 所示。

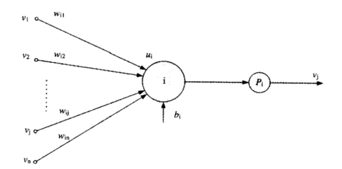


图 14-9 Boltzmann 机神经元结构

任意一个神经元的数学模型表征如下：

$$u_i=\sum_{j}^{n}w_{ij}v_j+b_i$$

首先输入进行加权组合，神经元输出取值范围为[0,1]，取 0 或 1 的概率依输入决定。神经元输出为 1 的概率：

$$P(v_i=1)=1/(1+\mathrm{e}^{-u_i/T})$$

神经元输出为 0 的概率：

$$P(v_i=0)=1-1/(1+\mathrm{e}^{-u_i/T})$$

显然，u_i 越大，则输出取值为 1 的概率越大；u_i 越小，则输出取值为 0 的概率越大。参数 T 称为网络的温度。T 升高时，网络概率随 u_i 的变化越来越不明显，而当 T 降低时，u_i 稍有变动则神经元的取值概率就会变动很大。当 $T\to 0$ 时，激励函数变为阶跃函数，每个神经元不再具有随机特性，这时 Boltzmann 机就退化为 Hopfield 网络。从这个意义上来说，Hopfield 网络实际上是 Boltzmann 机的一个特例。

14.2.2 模拟退火算法

模拟退火算法（Simulated Annealing，SA）是从物理和化学中的退火过程类推而来的。它由 Metropolis 算法和退火过程（Annealing Procedure，AP）组成。Metropolis 算法由 Metropolis 于 1953 年提出，这是一种改进的 MonteCarlo 算法，该方法早期用于在给定温

度下原子达到平衡状态的随机仿真计算。“退火”是物理学术语，意为对物体加温后再冷却的处理过程。

模拟退火算法的思路是：首先在高温下进行搜索，此时各个状态出现的几率相差不大，可以很快进入“热平衡状态”，这时进行的是粗搜索，大致找到低能区域；此后，随着温度的降低，各状态出现的概率的差距逐渐拉大，搜索精度不断提高，从而越来越准确地找到网络能量函数的全局极小点。

模拟退火算法的最初目的是寻找代表复杂系统的代价函数的全局最小值。这种方法为解决复杂的曲面最优问题提供了一个强有力的工具。在此算法过程中，系统的误差或者能量函数绝大部分时间在下降，但不是一直下降，即误差或者能量函数向着减小的方向变化，但有时也朝着增大的方向变化，这样就可以跳出局部极小点，向全局最小点收敛。

模拟退火算法的大致步骤如下。

① 随机给定初始状态，设定合理的退火策略。（选择各参数值、初始温度 T_0、降温规律等）

② 令 $x' = x + \Delta x$（Δx 为小的均匀分布的随机扰动），计算 $\Delta E = E(x') - E(x)$。

③ 若 $\Delta E < 0$，则接受 x' 为新的状态，否则以概率 $P = \exp(-\Delta E / (kT))$ 接受 x'，其中 k 为玻尔兹曼常数。具体做法是产生 0 到 1 之间的随机数 a，若 $P > a$ 则接受 x'，否则拒绝 x'，系统仍停留在状态 x。

④ 重复步骤②、③直到系统达到平衡状态。

⑤ 按第①步中给定的规律降温，在新的温度下重新执行②~④步，直到 T=0 或者达到某一预定的低温。

由以上步骤可以看出，$\Delta E > 0$ 时仍然有一定概率（T 越大概率越大）接受 x'，因而可以跳出局部极小点。理论上说，温度 T 的下降应该不快于：

$$T(t) = T_0 / (1 + \ln t),\ t = 1,2,3...$$

其中 T_0 为起始高温，t 为时间变量。常用的公式是 $T(t) = \alpha T_0 (t-1)$，其中 $0.85 \leqslant \alpha \leqslant 0.98$。

在最优化问题的计算中，模拟退火算法具有跳出局部最优陷阱的能力，因此被 Ackley、Hinton、Sejnowski 等人用做 Boltzmann 机的学习算法，从而使 Boltzmann 机克服了 Hopfield 网络经常收敛到局部最优点的缺陷。在 Boltzmann 机中，即使网络系统落入了局部最优的陷阱，经过一段时间之后，它还能够跳出来，使系统最终收敛到全局最优点。

模拟退火算法在求解规模较大的问题时，还存在收敛速度慢的缺点。为此，人们对模拟退火算法进行了各种改进，包括并行模拟退火算法、快速模拟退火算法等，这里就不再一一赘述。

14.2.3 Boltzmann 机的工作原理

同 Hopfield 网络一样，Boltzmann 机采用能量函数作为描述其网络状态的函数，其能量函数定义如下：

$$E = -\frac{1}{2}\sum_{i,j} w_{ij} v_i v_j$$

将 Boltzmann 机视为一个动力系统，则网络温度以某种方式下降到一个特定值时，系统将趋于一个稳定平衡态，此时也就对应着能量函数的极小值。在应用中，将需要求解的优化问题的目标函数与网络的能量函数相对应，就能够通过网络的稳定状态找到优化目标的极小值。

网络学习的目的是通过给出一组学习样本，经学习后得到 Boltzmann 机各种神经元之间的连接权值 w_{ij}。

Boltzmann 机采用的学习算法为模拟退火算法，其学习步骤可归纳如下。

① 随机设定网络的连接权值 $w_{ij}(0)$ 及初始高温。

② 按照已知的概率 $P(x_0)$，依次给定学习样本。在样本的约束下，按照模拟退火程度运行网络，直至达到平衡状态，统计出 P_{ij}。在无约束条件下，按同样的步骤和同样的次数运行网络，统计出 P_{ij}'。

③ 按下述公式修改权值：

$$w_{ij}(k+1) = w_{ij}(k) + \eta(P_{ij} - P_{ij}), \eta > 0$$

④ 重复上述步骤，直到 $P_{ij} - P_{ij}'$ 小于一定的容限。

与 Hopfield 网络的动力学演化相比较，Boltzmann 机处于某一状态的概率是与温度 T 有关的。

（1）网络温度 T 很高时，各状态出现的概率差异比较小，因而网络停留在全局极小点的概率并不比停留在局部极小点的概率高很多。在另一方面可以理解为，高的网络温度可以保证，网络并不会陷入某一个局部极小点而走不出来。

（2）网络温度 T 很低时，各状态出现的概率差异变得很大，因而网络一旦陷入某一个局部极小点后，要跳出该局部极小点的可能性是比较小的。但是，这基本上能够保证网络到达全局极小点时，跳出的可能性小。

（3）网络温度 $T \to 0$ 时，跳出局部极小点的概率趋于无穷小，这也保证网络最终状态能够稳定在全局极小点。

在 Hopfield 网络中，每一种约束相当于一种初始条件，不同的约束造成不同的初始条件可能使网络陷入不同的局部极小点。而相比较而言，Boltzmann 机则采用一种概率式的搜索方式，随着温度的降低，网络所处状态的概率集中于一个低能量状态的子集。

由于要进行多次模拟退火过程，因此 Boltzmann 机的学习计算量大，经平均场近似改善后，速度可以提高 1 到 2 个数量级，具体算法这里不再叙述。

14.3 基于遗传算法的神经网络优化

遗传算法（Genetic Algorithm，GA）是一种基于自然选择和基因遗传学原理的优化搜索方法。它将“优胜劣汰，适者生存”的生物进化原理引入待优化参数形成的编码串群体

中，按照一定的适配值函数及一系列遗传操作对各个体进行筛选，从而使适配值高的个体被保留下来，组成新的群体，新群体中各个体适应度不断提高，直至满足一定的极限条件。此时，群体中适配值最高的个体即为待优化参数的最优解。正是由于遗传算法独有的工作原理，使它能够在复杂空间进行全局优化搜索，并且具有较强的鲁棒性。

遗传算法应用于神经网络的一个方面是用来优化神经网络（ANN）的结构，另一个方面是用来学习 ANN 的权值，也就是用遗传算法取代一些传统的学习算法。

14.3.1 遗传算法介绍

遗传算法通过模拟自然环境中的遗传和进化过程，从而形成一种全局自适应优化概率搜索算法。

对于一个求函数极大值的优化问题，一般可以描述为下述数学规划模型：

$$\begin{cases} \max & f(\boldsymbol{X}) \\ s.t. & \boldsymbol{X} \in \boldsymbol{R} \\ & \boldsymbol{R} \subseteq \boldsymbol{U} \end{cases}$$

其中，$\boldsymbol{X}=[x_1,x_2,...,x_n]^T$ 为决策变量，$f(\boldsymbol{X})$ 为目标函数，第 2、3 式为约束条件，$\boldsymbol{U}$ 是基本空间，$\boldsymbol{R}$ 是 $\boldsymbol{U}$ 的一个子集。满足约束条件的解 $\boldsymbol{X}$ 称为可行解，集合 $\boldsymbol{R}$ 表示由所有满足约束条件的解组成的集合，称为可行解集合。

对于以上最优化问题，目标函数和约束条件种类繁多，各种不同的形式对应的解法也大不相同。常用的寻找最优解的方法有 3 种，分别是枚举法、启发式算法和搜索算法。

（1）枚举法：枚举出可行解集合内的所有可行解，求出精确最优解。这种方法只能对有限的可行解集合使用，并且效率较低。

（2）启发式算法：通过寻求一种能产生可行解的启发性规则，从而找到一个最优解或者近似最优解。这种方法对不同的问题需要找出其特定的启发式规则，应用时也有一定的难度。

（3）搜索算法：寻求一种搜索算法，此算法在可行解集合的一个子集内进行搜索操作，已找到问题的最优或近似最优解。例如梯度搜索算法，这种方法如果结合一些启发式知识，常常可以取得一种好的平衡。

在遗传算法中，将 n 维决策向量 $\boldsymbol{X}=[x_1,x_2,...,x_n]^T$ 用 n 个记号 $X_i(i=1,2,...,n)$ 所组成的符号串 $\boldsymbol{X}$ 表示：

$$\boldsymbol{X}=X_1X_2,...,X_n \Rightarrow \boldsymbol{X}=[x_1,x_2,...,x_n]^T$$

把每一个 X_i 看做一个遗传基因，它的所有的可能取值称为等位基因。这样 $\boldsymbol{X}$ 就可看做由 n 个遗传基因组成的一个染色体。一般情况下，染色体的长度 n 是固定的，但对一些问题，n 也可以是变化的。根据不同的情况，这里的等位基因可以是一组整数，也可以是某一范围内的实数值，或者纯粹的一个记号。最简单的等位基因是由 0 和 1 这两个整数组成的，相应的染色体就可以表示为一个二进制符号串。这种编码形式的排列形式 $\boldsymbol{X}$ 就是

个体的基因型，与之对应的 X 值就是个体的表现型。个体的适应度与其对应的个体表现型 X 的目标函数值相关联，X 越接近于目标函数的最优点，其适应度越大；反之，其适应度越小。

遗传算法中，决策变量 X 组成了问题的解空间。对问题最优解的搜索是通过对染色体 X 的搜索过程来进行的，从而所有的染色体 X 就组成了问题的搜索空间。遗传算法的运算过程也是一个反复迭代过程，第 t 代群体记做 $P(t)$，经过一代遗传和进化后，得到第 $t+1$ 代群体，它们也是由多个个体组成的集合，记做 $P(t+1)$。

遗传算法中最优解的搜索过程也是模仿生物的进化过程，通过染色体之间的交叉和变异来完成。通过所谓的遗传算子（Genetic Operators）作用于群体 $P(t)$中，进行遗传操作，从而得到新一代群体 $P(t+1)$。

（1）选择（Selection）。根据个体的适应度，按照一定的规则或方法，从第 t 代群体 $P(t)$ 中选择出一些优良的个体遗传到下一代群体 $P(t+1)$中。

（2）交叉（Crossover）。将群体 $P(t)$内的个体随机搭配成对，对每一对个体，以某个概率（称为交叉概率，crossover rate）交换他们之间的染色体。

（3）变异（Mutation）。对群体 $P(t)$中的每一个个体，以某一概率（称为变异概率，Mutation Rate）将某一个或某一些基因座上的基因值改为其他的等位基因。

遗传算法的运算过程如图 14-10 所示。

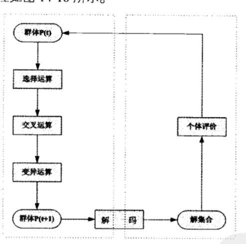


图 14-10 遗传算法流程

由图中可以看出，使用上述三种遗传算子（选择算子、交叉算子和变异算子）的遗传算法的主要运算过程如下所述。

① 初始化。设置进化代数计数器 t=0；设置最大进化代数 T；随机生成 M 个个体作为初始群体 $P(0)$。

② 个体评价。计算群体 $P(t)$中个体的适应度。

③ 选择运算。将选择算子作用于群体。

④ 交叉运算。将交叉算子作用于群体。

⑤ 变异运算。将变异算子作用于群体。群体 $P(t)$经过选择、交叉、变异运算之后得到下一代群体 $P(t+1)$。

⑥ 终止条件判断。若 $t \leqslant T$，则：$t=t+1$，转到步骤②；若 $t>T$，则以进化过程中得到的具有最大适应度的个体作为最优解输出，终止计算。

相对于其他各种优化算法，如单纯形法、梯度法、动态规划法、分支定界法等，遗传算法是一类可用于复杂系统优化计算的稳健的搜索算法，其特点如下。

（1）以决策变量的编码作为运算对象。

（2）直接以目标函数值作为搜索信息。

（3）同时使用多个搜索点的搜索信息。

（4）使用概率搜索技术。

14.3.2 基于遗传算法的神经网络优化算法

设有三层 BP 网络，I_i 为输入层中第 i 个结点的输出；H_i 为隐含层中第 i 个结点的输出；O_i 为输出层中第 i 个结点的输出；WIH_{ij} 为输入层中第 i 个结点与隐含层第 j 个结点的连接权值；WHO_{ji} 为隐含层中第 j 个结点与输出层第 i 个结点的连接权值。遗传算法学习 BP 网络的步骤如下。

① 初始化种群 P，包括交叉规模、交叉概率 Pc、突变概率 Pm 以及对任一 WIH_{ij} 和 WHO_{ji} 初始化；在编码中，采用实数进行编码，初始种群取值为 30。

② 计算每一个个体评价函数，并将其排序。可按下式概率值选择网络个体：

$$p_s = \frac{f_i}{\sum_{i=1}^{N} f_i}$$

其中 f_i 为个体 i 的适配值，可用误差平方和 E 来衡量，即：

$$f(i) = \frac{1}{E(i)}$$

$$E(i) = \sum_{p}\sum_{k}(V_k - T_k)^2$$

其中 $i=1$，……，N，表示染色体数；$k=1$，……，4 为输出层节点数；$p=1$，……，5 为学习样本数；T_k 为教师信号。

③ 以概率 Pc 对个体 G_i 和 G_{i+1} 交叉操作产生新个体 G_i' 和 G_{i+1}'，没有进行交叉操作的个体直接进行复制。

④ 利用概率 Pm 突变产生 G_j 的新个体 G_j'。

⑤ 将新个体插入到种群 P 中，并计算新个体的评价函数。

⑥ 如果找到了满意的个体，则结束，否则转至步骤③。达到所要求的性能指标后，将最终群体中的最优个体解码即可得到优化后的网络连接权值。

理论分析和实验结果表明，遗传算法作为一种新型的全局优化搜索算法，将它用于神经网络权值的训练学习能得到较好的结果，它能克服 BP 算法中学习效率低、收敛速度慢、容易陷入局部最优等缺点，是神经网络权值训练学习的有效方法。

遗传算法适合于处理规模较大的并行问题，因此，当网络的结构较庞大时，只要选择了合适的控制参数，就能够更充分地发挥出其收敛速度快，不至于陷入局部最小的优点。

14.3.3 遗传算法优化实例

遗传算法是人工智能算法中比较具有代表性的一种算法，本节对遗传算法应用进行介绍。神经网络工具箱中没有有关遗传算法的内容，运行本节中的实例需要安装本书附带光盘中的 GAOT 工具箱，此工具箱为第三方开放工具箱。

GAOT 工具箱的安装过程同 SVM-KM 工具箱类似，执行步骤如下。

① 将 gaot.工具箱压缩包解压到期望的文件夹，最好选择 MATLAB 安装目录下的 toolbox 文件夹。

② 在 MATLAB 命令窗口中选择“File”菜单下的“Set Path”选项。

③ 单击“Add with Subfolders”，在其中选择解压缩后的 gaot 文件夹，将其添加到 MATLAB 配置目录中。

④ 单击“Save”按钮保存配置，再单击“Close”按钮，即完成此工具箱的安装。

此工具箱中主要的函数包括 initializega 函数和 ga 函数，其中 ga 函数是遗传算法的应用函数。其调用格式为：

```
[x, endPop]=ga(bounds, evalFN, evalOps, startPop, opts, termFN, termOps,
selectFN, selectOps, xOverFNs, xOverOps, mutFNs, mutOps)
```

其中，函数的输出参数如下：

- x 为遗传运算过程中得到的最优解；
- endPop 为遗传运算得到的最终种群。

输入参数如下：

- bounds 为变量取值范围；
- evalFN 为定义优化函数的 M 文件名称；
- evalOps 为传递给优化函数的参数，通常为空；
- startPop 为初始种群；
- opts 为一个向量[epsilon prob_ops display]，其中 epsilon 为求解精度，prob_ops 为 0 或 1，决定遗传算法是否采用概率搜索方式，display 确定是否显示；
- termFN 为终止函数，通常设置为['maxGenTerm']；
- termOps 为传递给 termFN 的参数，为算法执行的次数；
- selectFN 为选择函数，通常为['normGeomSelect']；
- selectOpts 为传递给 selectFN 的参数，通常为[0.08]；
- xOverFNS 为交叉函数，通常可选['arithXover heuristicXover simpleXover']之一；

- xOverOps 为传递给交叉函数的参数，根据交叉函数的不同，通常对应[2 0;2 3;2 0]；
- mutFNs 为变异函数，通常可选['boundaryMutation multiNonUnifMutation nonUnifMutation unifMutation']之一；
- mutOps 为传递给变异函数的参数向量，第一个元素为交叉变异个数，第二个元素则为算法执行次数。

工具箱安装完毕之后，就可以进入下面的实例操作。

【例 14-3】 遗传算法优化实例。给定一维函数：$f(x)=x+10\sin(5x)+7\cos(4x)$，应用遗传算法对此函数在[0,9]取值区间内进行最大值搜索。

解：① 清空数据空间。在 MATLAB 命令空间中输入命令：

```
close all
clc
```

接下来绘制函数图形，并锁定画面。输入命令：

```
fplot('x + 10*sin(5*x)+7*cos(4*x)',[0 9])
hold on
```

绘出函数曲线如图 14-11 所示。

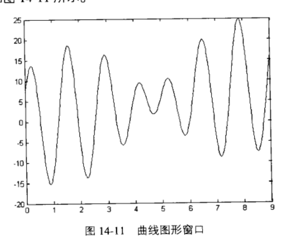


图 14-11 曲线图形窗口

② 初始化遗传算法。输入命令：

```
initPop=initializega(20,[0 9],'gademo1eval1');
```

此处，initializega 函数为遗传算法的初始化函数，其调用格式如下：

```
[pop] = initializega(num, bounds, evalFN)
```

initializega 函数执行的结果是返回一个随机初始点及其函数值构成的二维矩阵 pop 作为遗传的初始种群。其中，初始点个数为 num，此处设定初始点个数为 20；取值范围由 bounds 确定，此处定为[0,9]；evalFN 为初始化函数，通常由 M 函数文件定义，这里的初始化函数定义在 gademo1eval1.m 文件中。

我们可以查看 gademoeval1.m 文件中的函数定义。输入命令：

```
clc
type gademo1eval1
```

从输出的内容中可以看到，此 M 文件中定义的函数正是：

$f(x)=x+10\sin(5x)+7\cos(4x)$

下面对 initializega 函数生成的初始点进行绘图显示，在原函数曲线上绘出初始点，以加号“+”表示。输入命令：

```
plot (initPop(:,1),initPop(:,2),'+')
pause;
```

绘出图形如图 14-12 所示。

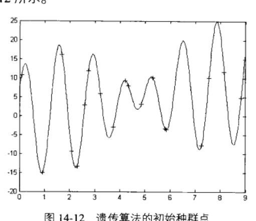


图 14-12 遗传算法的初始种群点

③ 调用 ga 函数进行遗传运算，首先进行 1 次遗传、交叉和变异，并查看算法执行的效果。输入命令：

```
[x endPop] = ga([0 9],...
'gademo1eval1',[],initPop,[1e-6 1 1],'maxGenTerm',1,...
'normGeomSelect',[0.08],['arithXover'],[2 0],'nonUnifMutation',[2 1 3]);
x,
```

对结果进行绘图显示，在原函数曲线上绘出 1 次遗传运算后的取值点，以空心圆圈表示。输入命令：

```
plot (endPop(:,1),endPop(:,2),'ro')
pause;
```

绘出图形如图 14-13 所示。可见经过 1 次遗传、交叉、变异运算之后，种群位置在函数曲线上发生了移动。

④ 调用遗传算法，进行 50 次遗传运算。输入命令：

```
[x endPop] = ga([0 9],...
'gademo1eval1',[],initPop,[1e-6 1 1],'maxGenTerm',50,...
```

```
'normGeomSelect',[0.08],['arithXover'],[2],'nonUnifMutation',[2 50 3]);
```

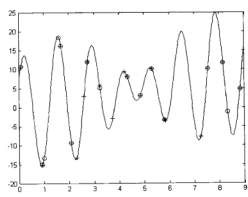


图 14-13　一次遗传运算后的种群点

在原函数曲线上绘出 50 次遗传运算后所取的点以星号“*”表示。输入命令：

```
plot (endPop(:,1),endPop(:,2),'b*')
hold off
```

绘出图形如图 14-14 所示。

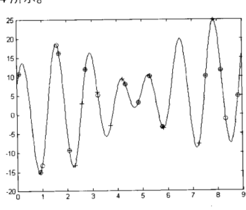


图 14-14　50 次遗传运算后的种群点

从图中可以看到，最终种群确实收敛到了函数的最大值点。

对最优解进行查看，输入命令：

```
x
```

输出结果为：

```
x =    7.8567   24.8554
```

此即遗传运算寻得的最优解，对应取值区间内优化函数的最大值。

上面通过 gaot 工具箱介绍了遗传算法应用于函数优化的基本过程，应用遗传算法对神经网络进行优化是神经网络技术中一个很有前景的发展方向，更多的实例可以参考其他资料。

14.4 小结

神经网络和人工智能技术发展了数十年，目前所包含的内容已经非常丰富，本章所述的神经网络优化方法就是一些比较有前景和应用价值的方法，不过在目前版本的 MATLAB 神经网络工具箱中没有对应的函数。本章的目的主要在于提出这些方法，因此不展开细致的分析，有兴趣的读者可以参考详细论述的书籍，做进一步的深入学习。



第 15 章　神经网络控制

自动控制理论已有 50 年的历史了。按照常规分类，自动控制分为两个分支，连续控制和离散控制。在连续控制中信号是时变的，并且由过去的模拟量闭环控制转为由计算机实施的数字控制。离散控制系统很大程度上已经趋于继电开关式控制（可编程控制器 PLC），系统实质上是在进行启动、停止、等待或监测一系列的离散事件。

尽管自动控制技术已经成功地在工业中得到了应用，并实现了降低产品费用、确保生产过程可靠性的目标，但是，大部分自动控制仍然局限在较小的范围内，局限于对某一个或几个物理参量的控制。自动控制的目标是实现全方位的系统或过程的最优调节，这不仅包括机器本身，还应考虑人的行为，首先要做的工作就是对系统进行准确的建模。然而，要建立一个完整的复杂系统的模型是很困难的，并且在很多情况下也不必要。

在自动控制的过程中，总代价成本（包括资源、能量、工作量等）也是一个非常关键的量。我们对系统的控制，总是需要对整个过程所有参量进行集中控制。然而，在集中控制这些参量时，往往会遇到棘手的问题，其中的问题包括：

1. 固有的不稳定性

系统或者整个过程可能是动态不稳定的。在这样的系统中，不改变控制的状态可能会导致系统性能的恶化。

2. 数据不完整或过多

大量观测数据可能有噪声，是不可信的，另一方面，数据也可能太多以致存在冗余，我们必须筛选出有用的数据。

3. 不可辨识的过程

仍然存在一些不可辨识的过程，这给控制造成了困难。

传统的控制理论建立在数学模型的基础上，甚至我们认为，必须找到系统的线性化模型才能够实现合理的控制。因为我们对非线性和时变系统，并没有很好的统一处理方法。这是制约自动控制的一个瓶颈问题。

然而，随着计算机的快速发展，不基于数学模型的控制也快速发展起来。比如将人工智能、计算机技术与自动控制结合起来，通过人工神经网络的自学习、自组织、容错性和并行性来实现系统控制，能够更好地处理系统的非线性特性，并进行多变量并行分布式处理，是非常具有优势的。

15.1 神经网络控制概述

神经网络在控制系统中的应用从 20 世纪 60 年代就已经开始了，最早的工作是由 Widrow 和 Hoff 做出的。20 世纪 80 年代以后，随着计算机与人工智能热潮的兴起，Neurocontrol 这个新名词即应运而生了。

神经网络相对于传统控制手段，具有以下优势。

（1）非线性特性。神经网络能够以任意精度实现非线性映射，从而可以适用于复杂的系统建模。

（2）并行分布式处理。神经网络具有并行结构，可以进行并行数据处理，同时具有更强的容错能力。

（3）学习与自适应能力。神经网络是一种时间序列处理的方法，应用过去的数据记录进行训练，因此便于进行在线自适应调节。

（4）多变量系统。神经网络可以处理多个输入信号，可以适用于多变量系统。

从目前来看，神经网络在非线性系统中的应用是非常重要的，因为目前并没有一个普遍可适用的理论来指导非线性系统的设计；而利用神经网络提供非线性系统的模型，基于理论方法和优化技术设计非线性控制器，则是一个比较有前途的方向。另一方面，神经网络可以应用于动态变化的系统，与控制理论中的自适应系统具有一定相似性。

15.1.1 监督式神经网络控制

对于受控系统的动态特性是未知或仅有部分是已知的情况，需要寻找一种支配系统行为的规律，以便系统能够被有效地控制。有些情况下，可能需要设计一种模仿人类作用的自动控制器，基于规则的专家控制和模糊控制是一种方法，而神经网络控制是另一种可选的方法，也称为监督式神经控制。监督式神经网络控制系统的结构如图 15-1 所示。

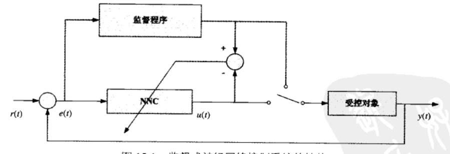


图 15-1 监督式神经网络控制系统的结构

图 15-1 所示的监督式神经网络控制系统包括以下几个组成模块：监督程序、NNC（可训练的神经网络控制器）、受控对象。其中，NNC 的输入就是由人接收的传感输入信息，而输出则应用于对系统的控制。监督式神经网络控制系统实现控制具有以下几个步骤。

① 通过传感器和传感信息处理，调用必要的和有用的控制信息。

② 选择神经网络类型、结构参数和学习算法等等，构造神经网络。

③ 训练神经网络控制器，实现输入与输出间的映射，从而进行正确的控制。训练过程可采用线性律、反馈线性化或解耦变换的非线性反馈作为监督程序，对网络进行训练。

15.1.2 直接逆模型神经网络控制

所谓的直接逆模型神经网络是指将受控系统的一个逆模型与受控系统串联，从而使系统在期望响应与受控系统输出之间得到一个相同的映射。此网络直接作为前馈控制器，并且受控系统的输出等于期望输出，此控制方案已经用于机器人控制。这种方法进行控制的效果很大程度上依赖于作为控制器的逆模型的精确程度。由于缺少反馈，所以这种方法的健壮性不够。不过，通过调整学习参数，这种方法的健壮性是可以提高的。

一种直接逆模型神经网络控制系统的结构如图 15-2 所示。

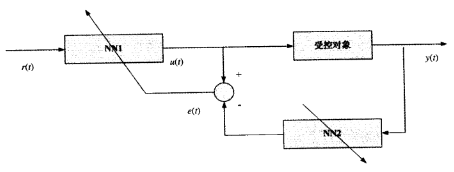


图 15-2 直接逆模型神经网络控制系统的结构

15.1.3 神经网络自适应控制

神经网络自适应控制与常规自适应控制是类似的，可以分为两类，分别是神经网络自校正控制（STC）和模型参考自适应控制（MRAC）。这两类控制系统之间存在一定的差别：自校正控制系统是根据受控系统的正或负逆模型辨识结果来对常规控制器的内部参数进行调节，从而满足系统给定的性能指标；而在模型参考自适应控制系统中，闭环控制系统的期望性能由一个稳定的参考模型给出，控制系统的目标是使得受控系统的输入 $y(t)$与参考模型的输出 $y^{T}(t)$近似匹配。

下面分别对其进行说明。

1．神经网络自校正控制

神经网络自校正控制又分为两种类型，分别是直接自校正控制和间接自校正控制。其中，直接自校正控制系统由一个常规控制器与一个识别器组成，后者具有很高的建模精度和识别能力，结构上与直接逆模型控制系统类似。间接自校正控制系统的结构如图 15-3 所示。

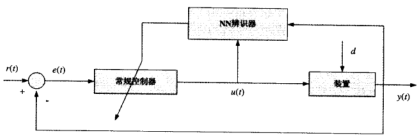


图 15-3 神经网络间接自校正控制系统的结构

图 15-3 中的控制系统由一个常规控制器与一个神经网络辨识器构成，能实现对受控装置的控制。受控对象可以以一个单变量非线性系统模型来表征：

$$y_{k+1} = f(y_k) + g(y_k)y_k$$

其中，$f(y_k)$ 与 $g(y_k)$ 均为非线性函数。y_{k+1} 为系统在 k+1 时刻的输出。

2．神经网络模型参考自适应控制

神经网络模型参考自适应系统的组成结构如图 15-4 所示。

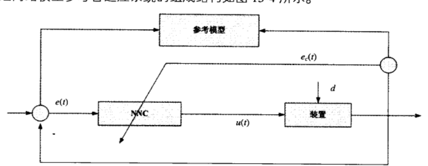


图 15-4 神经网络模型参考自适应控制系统的结构

如图 15-4 所示，系统控制的目标是维持受控对象输出与参考模型输出之间的差：

$$e_c(t) = y(t) - y^m(t)$$

由于在反向传播控制中需要知道受控系统的数学模型，因此这种控制系统的学习和修正还存在一些问题。

15.1.4 神经网络内模控制

在常规内模控制系统中，通常是采用受控系统的正逆模型作为反馈回路中的单元，这种控制方式经过了比较全面的检测，其不仅可以用于健壮性和稳定性分析，而且是一种新的重要的非线性系统控制方法。其结构如图 15-5 所示。

图 15-5 所示系统由滤波器、逆模型神经网络控制器 NN1、系统模型 NN2，以及受控装置构成。其中系统模型 NN2 是基于神经网络的一个具有系统特性的正向模型，它是与



受控装置并行设置的，而反馈系统由系统输出与系统模型输出之间的差得到，并且在控制器 NN1 中进行处理，此控制器是根据系统的逆模型给出的。滤波器在系统中的用途是使系统满足必要的健壮性，同时实现闭环系统跟踪响应。

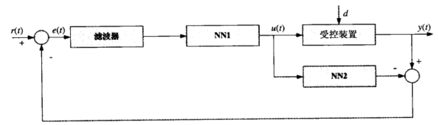


图 15-5　神经网络内模控制系统的结构

15.1.5　神经网络预测控制

预测控制是基于模型的一类控制，产生于 20 世纪 70 年代，其特点是具有预测模型、滚动优化以及反馈校正等处理过程。经证明，这种控制方法对于非线性系统能够产生比较符合期望的稳定性。一种神经网络预测控制系统的结构如图 15-6 所示。

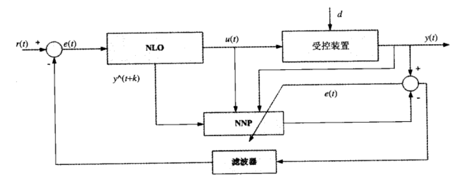


图 15-6　神经网络预测控制系统的结构

图 15-6 所示，系统包含神经网络预测器 NNP、非线性优化器 NLO、受控装置、滤波器 4 个组成部分。其中 NNP 用来预测受控对象在一定范围内的未来响应，其公式可以写为：

$$y(t+j\,|\,t),\ j=N_1, N_1+1, \ldots, N_2$$

其中，N_1，N_2 分别是输出预测的最小和最大级别。

如果在（t+j）时刻系统的预测误差定义为：

$$e(t+j)=r(t+j)-y(t+j\,|\,t)$$

则非线性优化器 NLO 将选择信号 $u(t)$，以便使得系统的二次性能判决函数 J 达到最小：

$$J=\sum_{j=N_1}^{N_2} e^2(t+j)+\sum_{j=1}^{N_2} \lambda_j \Delta^2 u(t+j-1)$$

其中，$\Delta u(t+j-1)=u(t+j-1)-u(t+j-2)$

15.1.6 神经网络自适应判断控制

神经网络自适应判断模型是由 Barto 等人提出，并由 anderson 等人发展的。这种方法应用强化学习原理，可以在缺少受控对象的期望输入的情况下，尤其是在系统模型未知或者部分未知而无法提供期望输入的情况下，实现对系统的控制。

这种控制系统包括两个网络：一个是自适应判断网络 AJN，另一个是控制选择网络 CSN。其结构如图 15-7 所示。

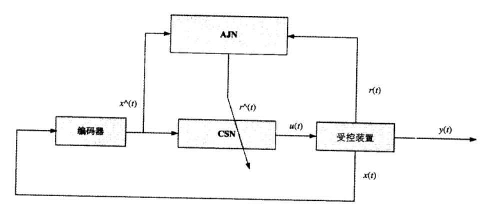


图 15-7 神经网络自适应控制系统的结构

其中，自适应判断网络 AJN 相当于强化学习需要的“导师”。它有两个作用：一是通过不断的赏罚与强化学习，使 AJN 成为一个数量的导师；二是通过学习，根据受控系统的当前和外部强化反馈信号 $r(t)$，产生一个强化信号，提供内部强化信号，从而判断当前控制作用的效果。

控制选择网络 CSN 是一个多层前馈神经网络控制器，它在内部强化信号的引导下进行学习，经过学习之后的 CSN 网络，应该能够根据当前系统编码的状态，选择给出下一个控制信号。

15.1.7 多层神经网络控制

多层神经网络控制器的结构如图 15-8 所示，它包含前馈控制器、常规控制、受控装置 3 个部分。此系统的前馈控制部分是由神经网络实现的，其训练目标在于使期望输出与实际装置的输出之间的偏差达到最小，而该误差作为反馈控制器的输入。在此过程中，反馈作用与前馈作用是分别考虑的。

在此情况下，多层神经网络控制器又分为间接结构、专用结构、通用结构等结构形式，这里不再赘述。

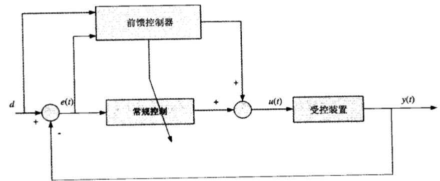


图 15-8　多层神经网络控制器的结构

15.1.8　分级神经网络控制

分级神经网络控制系统的结构如图 15-9 所示，包括 4 个组成部分：神经网络 I、神经网络 II、常规控制器，以及受控装置。其中，d 为受控装置的期望输出，它作为输入传递给神经网络 I、神经网络 II 以及常规控制器，$u(t)$为受控装置的控制输入，$y(t)$为受控装置的实际输出。$u^*(t)$是由神经网络 II 的输出，而 $y^*(t)$则是神经网络 I 的输出。

此控制系统可看做由三部分构成。

第一部分是一个常规外反馈回路。反馈控制是以期望装置输出 d 和由传感器测量的实际装置输出 $y(t)$之间的误差 $e(t)$为基础的。通常，常规控制器为一个比例微分控制器。

第二部分是神经网络 I 通道部分。它可以构成一个受控对象的动力学内模型，用于监控受控装置的输入 $u(t)$和输出 $y(t)$，并且学习受控对象的动力学特性。当接收到一个受控装置输入 $u(t)$时，通过与实际系统输出 $y(t)$进行比较，神经网络 I 计算出一个受控装置近似输出 $y^*(t)$。从这个意义上看，这部分起到的是一个系统动态特性辨识器的作用。

第三部分是神经网络 II 通道部分。它通过系统期望输出 d 与受控装置输入 $u(t)$，计算得到一个合适的受控装置输入分量 $u^*(t)$。

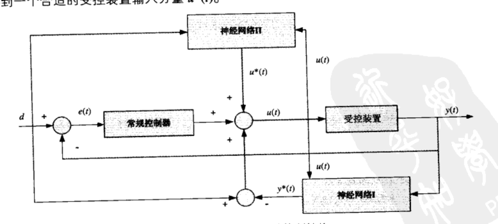


图 15-9　分级神经网络控制结构

在此控制系统的学习阶段，神经网络 I 学习系统的动力学特性，而神经网络 II 学习系统的逆动力学特性。随着学习进行，内反馈逐渐替代外反馈的作用，成为主控制器。然后当学习进一步进行时，逆动力学部分将取代内反馈控制。最后，此装置主要由前馈控制器进行控制，因为装置的输出误差与内反馈一起几乎不复存在，因此可以提供处理随机扰动的快速控制。系统控制的最后效果与前馈控制效果是接近的。

实际上，完全可以将分级神经网络划分为两个系统：基于正向动力学器辨识器的系统和基于逆向动力学辨识器的系统，单独进行应用。

前面介绍了各种类型的神经网络控制系统。可以看到，由于神经网络具有全局逼近的能力，因此在非线性系统建模和常规非线性控制器得到了很好的应用。

下面的几节将会介绍三种神经网络控制系统的应用实例，同时每一节都有对其基本控制原理的描述，并有实际控制效果的仿真演示。它们分别是：

- 神经网络模型预测控制；
- 神经网络反馈线性化控制；
- 神经网络模型参考控制。

神经网络控制通常包含两个步骤，分别是：

- 系统辨识；
- 控制系统设计。

在系统辨识阶段，主要任务是建立受控装置的神经网络模型；在控制设计阶段，则使用神经网络模型来设计或训练控制器网络。对于上述三种网络，系统辨识阶段是完全相同的，而控制设计阶段则各不相同。

（1）对于模型预测控制，需要应用受控系统模型对系统未来的行为进行预测，同时应用最优化算法对最优化系统性能的控制输入进行选择。

（2）对于反馈线性化控制，控制器只是简单地对受控系统模型进行调整。

（3）对于模型参考预测，控制器是一个经过训练的用来对装置进行控制的神经网络，它跟踪一个参考模型，而神经网络系统状态模型则用来对控制器训练过程进行辅助。

15.2 神经网络模型预测控制

神经网络预测控制模型使用非线性神经网络模型来预测未来模型的性能。在神经网络工具箱中，受控装置的控制输入信号由控制器产生，而控制输入在未来一段指定的时间内将模型性能最优化。模型预测的第一步是要建立神经网络模型，对系统进行辨识；第二步，是使用控制器来预测未来神经网络的性能。

如前所述，这种神经网络控制方法的优点是，对于非线性系统能够产生比较符合期望的稳定性。其主要特点如下所述。

（1）控制器应用神经网络模型，对系统对所有可能控制信号的反应进行预测。

（2）神经网络系统模型的训练属于离线训练方式，可以选择任意的学习批处理算法。

（3）选择某一种优化算法，对系统进行最优化计算。

（4）控制过程中需要大量地再现计算数据。

15.2.1 系统辨识

近几年来，神经网络在复杂动力学系统控制和辨识中的应用受到了普遍的关注，利用神经网络预测可以解决很多非线性和非确定性的控制问题。

利用神经网络进行模型预测的第一步就是训练神经网络，以模拟系统的动力学特性。系统输出与神经网络输出之间的预测误差用来提供神经网络的训练信号。整个训练过程可以用图 15-10 表示。其中 $\boldsymbol{u}$ 为系统输入，$\boldsymbol{y}_p$ 为装置输出，神经网络模型以 $\boldsymbol{u}$ 和 $\boldsymbol{y}_p$ 作为训练输入样本向量，得到的网络输出 $\boldsymbol{y}_m$ 与装置输出 $\boldsymbol{y}_p$ 相减作为网络模型预测误差反馈给网络，从而对网络模型进行训练，完成对装置的辨识。

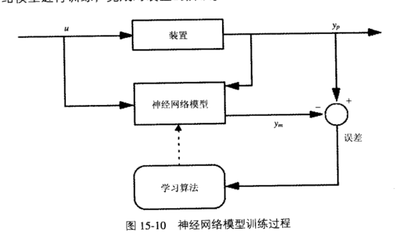


图 15-10 神经网络模型训练过程

图 15-10 中的神经网络模型利用当前输入和模型输出来预测网络的未来输出，其模型结构如图 15-11 所示。

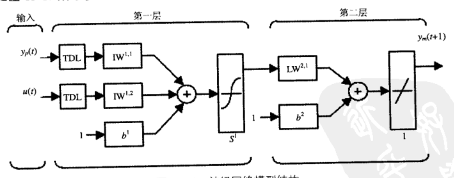


图 15-11 神经网络模型结构

从图 15-11 中可以看到，预测网络模型通常采用 BP 前向网络，t 时刻的输入训练样本为装置输入 $u(t)$和装置输出 $y_p(t)$，通过一定的时延之后通过第一层（sigmoid 层），以及第二层（线性层），获得 $t+1$ 时刻的网络输出 $y_m(t+1)$，同时计算误差，并采用误差反向传播

的方式对网络进行训练。

15.2.2 预测控制

利用神经网络进行模型预测控制，其原理基于所谓的水平后退方法（Receding Horizon Technique，RHT）。当神经网络获得输入后，在指定的时间内对模型响应进行预测。预测过程使用数字最优化程序来确定控制信号，其性能函数定义如下：

$$J=\sum_{N_1}^{N_2}(y_r(t+j)-y_m(t+j))^2+\rho\sum_{j=1}^{N_u}(u(t+j-1)-u(t+j-2))^2$$

其中，$N_1,N_2,\ldots,N_u$ 是计算跟踪误差和控制增益的范围，y_r 是期望响应，y_m 是网络模型响应，变量 u 是实验控制信号，即受控装置的控制输入，ρ 是控制增益平方和分布的因子。

网络模型预测控制过程如图 15-12 所示。

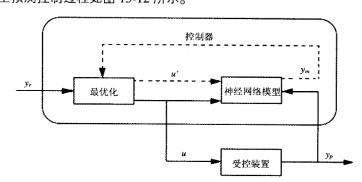


图 15-12 神经网络预测控制模型

如图 15-12 所示，控制器是由神经网络模型和最优化模块组成的，最优化模块通常利用动态网络结构实现最优化算法，以确定实验控制信号 u 的值。其中，期望响应 y_r 为系统输入，最优化模块完成指标函数的优化，产生控制信号 u，输出至神经网络模型和受控装置，而网络模型响应 y_m 反馈到最优化模块作为优化计算的一个输入；另一方面，控制信号 u 输出至受控装置，得到系统输出 y_p。

15.2.3 预测控制的 Simulink 实例

在 MATLAB R2008b 中，有一个利用神经网络进行模型预测控制的实例，该实例为研究催化剂的连续搅拌反应器（CSTR），此实例是基于 Simulink 环境的（Simulink 是 MATLAB 环境下用于系统建模、动态分析的软件包，关于其详细介绍请参考第 18 章）。下面对此搅拌系统实例进行介绍。

【例 15-1】 预测控制实例。如图 15-13 所示，要控制的系统为生产某种催化剂产品溶液的搅拌系统。此催化剂由同一溶质而不同浓度的两种溶液混合而成，其中浓度高的称

为浓缩液，浓度低的称为稀释液。这两种溶液从上而下注入一个容器，容器内部的螺旋桨对混合液进行搅拌，以达到充分混合，获得所需特定浓度的产品液，然后输出。

此系统的动力学模型可以表示为如下微分方程组：

图 15-13　连续搅拌反应器示意图

$$\frac{\mathrm{d}h(t)}{\mathrm{d}t}=w_1(t)+w_2(t)-0.2\sqrt{h(t)}$$

$$\frac{\mathrm{d}C_b(t)}{\mathrm{d}t}=(C_{b1}-C_b(t))\frac{w_1(t)}{h(t)}+(C_{b2}-C_b(t))\frac{w_2(t)}{h(t)}-\frac{k_1C_b(t)}{(1+k_2C_b(t))^2}$$

其中，$h(t)$为液面高度，$C_b(t)$为产品输出浓度，$w_1(t)$为浓缩液的流速，$w_2(t)$为稀释液的输入流速，C_{b1}为浓缩液的输入浓度，C_{b2}为稀释液的输入浓度。设定：$C_{b1}=24$，$C_{b2}=0.1$；$k_1=k_2=1$ 为消耗率。

问题是设计一个神经网络预测控制器，以便调节稀释液的流速 $w_2(t)$，保证产品具有合适的浓度。为简化演示过程，可以假定 $w_1(t)=0.1$。其中，液面高度 $h(t)$ 为不受控量。

解： ① 建立模型。在 MATLAB 命令空间中输入命令：

```
predcstr
```

此命令将直接启动 Simulink 界面，同时弹出如图 15-14 所示窗口，其中包含了神经网络预测控制模型。

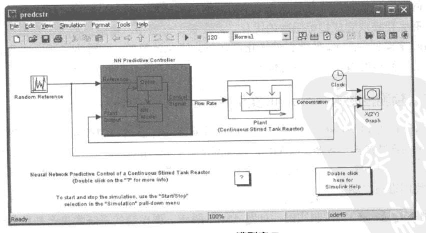


图 15-14　predcstr 模型窗口

图 15-14 所示的窗口中包含了下述几个模块：神经网络预测模块（NN Predictive Controller）、CSTR 模块（Continuous Stirred Tank Reactor）、随机信号发生器模块（Random Reference）、图形显示模块（Graph）。其中前两个模块都是 MATLAB 提供的 Simulink 模块，双击它们可以查看其中的具体内容，例如双击 CSTR 模块图标，将弹出如图 15-15 所示的窗口。此窗口显示了 CSTR 系统的构成，其中内容不再详细介绍。

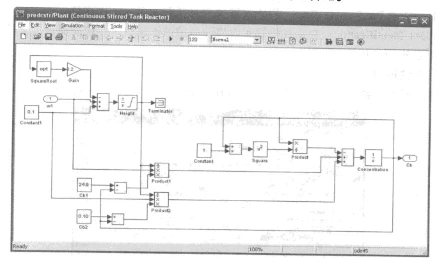


图 15-15　受控系统构成示意图

NN Predictive Controller 模块是在神经网络工具箱中生成和复制过来的，它的输出信号与 CSTR 模块连接，作为控制信号；而 CSTR 的输出信号又反馈输出到 NN Predictive Controller 模块的输入端，作为最优化的输入参考信号。

双击 NN Predictive Controller 模块图标，将弹出如图 15-16 所示的新窗口。

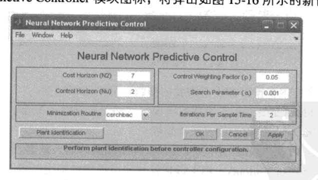


图 15-16　Predictive Control 模型窗口

通过此窗口，可以对该模型预测控制器进行设计。在这个窗口可以改变的控制参数包括：Cost Horizon N2、Control Horizon Nu、Control Weighting Factor ρ、Search Parameter α 等。其中各参数的解释分别如下。

- Cost Horizon N2：此参数为用于最小化预测误差的时间步数。

- Control Horizon Nu：此参数为用于最小化控制增量的时间步数。
- Control Weighting Factor ρ：此参数为控制权值因子，用于在性能函数中乘以控制增量的平方和。
- Search Parameter α：此参数为搜索参数，用来设定一维搜索的停止时间。
- Minimization Routine：用于选择最小化搜索程序。
- Iterations Per Sample Time：用于设置每个采样时间内的迭代次数。
- Plant Identification：单击此按钮可以打开系统辨识窗口，在控制器使用之前，先对系统进行辨识。

② 系统辨识。

设定好模型参数的值，单击“Plant Identification”按钮，将打开如图 15-17 所示的窗口。

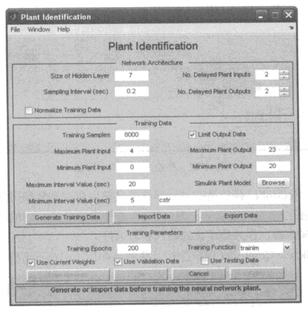


图 15-17 “Plant Identification”窗口

此窗口中包含有很多模型辨识所需要的参数，分别解释如下。

- Size of Hidden Layer：用于设置系统模型网络隐层神经元数目。
- Sampling Interval：用于设置采样间隔。
- No. Delayed Plant Inputs：用于设置系统网络模型的输入延迟。
- No. Delayed Plant Outputs：用于设置系统网络模型的输出延迟。
- Normalize Training Data：用于设置是否采用 premnmx 函数对数据进行归一化处理。
- Training Samples：用于设置训练样本的数目。
- Maximum Plant Input：用于设置随机输入的波峰值。
- Minimum Plant Input：用于设置随机输入的波谷值。
- Maximum Interval Value：用于设置随机输入的最大间隔。

- Minimum Interval Value：用于设置随机输入的最小间隔。
- Limit Output Data：用于选择系统输出是否有界。
- Maximum Plant Output：用于设置输出的最大值。
- Minimum Plant Output：用于设置输出的最小值。
- Simulink Plant Model：选择产生数据的 Simulink 系统模型。
- Generate Training Data：产生用于训练的数据。
- Import Data：从工作空间或者一个文件中导入数据。
- Export Data：将训练数据导出到工作空间或文件中。
- Training Epochs：设置训练迭代次数。
- Training Function：设置训练函数。

在此窗口中，单击“Generate Training Data”按钮，会产生一系列随机数据作为输入 Simulink 系统模型的训练样本数据，这些数据输入 Simulink 系统模型后得到输出数据，所绘出图形如图 15-18 所示。

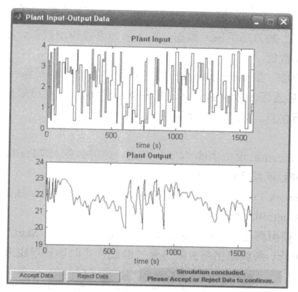


图 15-18 生成的随机训练样本数据

在图 15-18 中，上半部分是输入模型的数据，而下半部分是模型输出的数据，单击“Accept Data”按钮，就接受了这些数据；而单击“Reject Data”按钮，就将丢弃这些数据，重新产生训练样本。

本例先单击“Accept Data”按钮，然后在“Plant Identification”窗口中单击“Train Network”按钮，即开始对系统网络模型的训练。训练过程根据所选择的训练算法（trainlm）进行计算。这是一种直接批处理训练的应用。此时将弹出两个图形窗口，分别为训练数据曲线和验证数据曲线窗口，其中训练数据曲线如图 15-19 所示，验证数据曲线如图 15-20 所示。

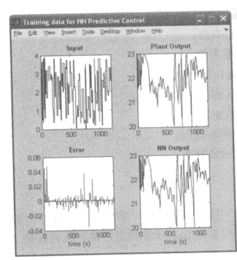


图 15-19 训练数据曲线

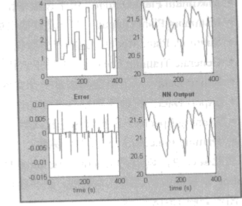


图 15-20 验证数据曲线

以上两图中，左上角为随机输入数据，右上角为系统输出数据，左下角为误差曲线，右下角则为网络输出数据。

接下来，可以单击“Train Network”按钮对网络重新进行训练，也可以选择“Erase Generated Data”然后重新生成新的数据，或者单击“OK”按钮接受目前的系统模型，并且对此闭环系统进行仿真，如下面的步骤所示。

③ 系统仿真。

单击 Plant Identification 窗口中的“OK”按钮，这将会在 NN Predictive Controller 模块中载入训练过后的神经网络系统模型。

在“Neural Network Predictive Controller”窗口中单击“OK”按钮，将控制器参数导入到 NN Predictive Controller 模块中。

回到 Simulink 模型界面下，在“Simulation”菜单下选择“Start”命令，就可以开始仿真。随着仿真的进行，系统输出与参考信号的曲线也实时地显示出来，如图 15-21 所示。

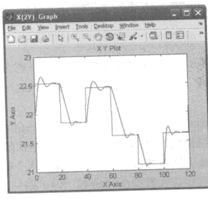


图 15-21 系统仿真结果

图 15-21 中，参考信号呈阶跃性变化，而系统的仿真输出则能够很好地跟踪此参考信号的变化，显示了神经网络模型预测控制的效果。

15.3 神经网络反馈线性化控制（NARMA-L2）

神经网络反馈线性化控制（又称为 NARMA-L2 控制），是通过反馈消除非线性，将非线性系统转换为线性系统来进行控制的神经网络控制方法。

当系统是伴随型特殊形式时，应用反馈线性化控制，当系统是伴随型模型估计时，应用 NARMA-L2 控制。NARMA-L2 控制系统具有下面的特点。

（1）计算量相对较小。

（2）神经网络系统模型的训练属于离线训练方式，训练方法上可以选择任意的学习批处理算法。

（3）控制系统中具有唯一的在线训练部分，即神经网络控制器的一个前向通路。

（4）能够应用反馈线性化控制的系统必须是伴随型，或能够应用伴随型模型估计。

本节首先介绍反馈线性化模型，以及如何利用神经网络进行系统辨识；接下来介绍如何利用经过辨识的神经网络模型来创建一个控制器，最后介绍 Simulink 提供的一个 NARMA-L2 控制系统，并对此系统进行说明。

15.3.1 NARMA-L2 系统辨识

同神经网络模型预测控制一样，NARMA-L2 控制的第一步也是对受控系统进行系统辨识。训练一个网络对系统的前向动力学特性进行复现，首先需要选择模型结构，一个用于表征一般离散非线性系统的标准模型是非线性自回归移动平均（NARMA）模型，其计算公式如下：

$$y(k+d)=N[y(k),y(k-1),\dots,y(k-n+1),u(k),u(k-1),\dots,u(k-n+1)]$$

其中，$u(k)$ 是系统输入，而 $y(k)$ 为系统输出。在系统辨识阶段，可以对网络进行训练以便对非线性函数 N 进行逼近。这其实也是利用 NN Predictive Controller 进行模型辨识的步骤。

如果希望网络输出能够跟随某些参考轨迹 $y(k+d)=y_r(k+d)$ 的话，就要创建一个下列形式的非线性控制器：

$$u(k)=G[y(k),y(k-1),\dots,y(k-n+1),y_r(k+d),u(k-1),\dots,u(k-m+1)]$$

这种控制器的问题在于，如果想要训练一个神经网络以便创建函数 G 对均方误差最小化的话，我们需要应用动态反向传播的方法，而这是一个非常缓慢的过程。由 Narendra 和 Mukhopadhyay 提出的一种解决方法是：利用进行模型来表征系统。此时，系统控制器就是基于 NARMA-L2 近似模型的。

$$\hat{y}(k+d)=f[y(k),y(k-1),\ldots,y(k-n+1),u(k-1),\ldots,u(k-m+1)]$$
$$+g[y(k),y(k-1),\ldots,y(k-n+1),u(k-1),\ldots,u(k-m+1)]u(k)$$

此模型是伴随形式的，其中下一个控制器输入 $u(k)$ 不包括在非线性之内。这种形式的优势在于可以控制输入，使得系统输出跟踪参考曲线 $y(k+d)=y_r(k+d)$。最终的控制器形式如下：

$$u(k)=\frac{y_r(k+d)-f[y(k),y(k-1),\ldots,y(k-n+1),u(k-1),\ldots,u(k-m+1)]}{g[y(k),y(k-1),\ldots,y(k-n+1),u(k-1),\ldots,u(k-n+1)]}$$

但是，直接使用上述方程还有一定的问题，因为首先需要通过同一时刻的输出 $y(k)$ 确定控制输入 $u(k)$。因此，通常采用下面的模型：

$$y(k+d)=f[y(k),y(k-1),\ldots,y(k-n+1),u(k-1),\ldots,u(k-n+1)]$$
$$+g[y(k),y(k-1),\ldots,y(k-n+1),u(k-1),\ldots,u(k-n+1)]u(k+1)$$

其中，$d\geqslant 2$。图 15-22 显示了这种神经网络系统模型的结构。

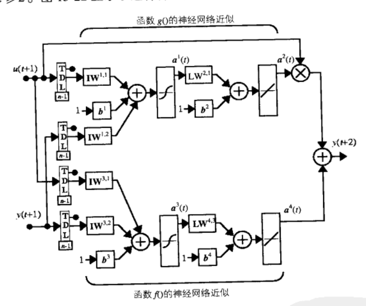


图 15-22　NARMA-L2 神经网络系统模型的结构

15.3.2 NARMA-L2 控制器

使用 NARMA-L2 模型，得到的控制器为：

$$u(k+1)=\frac{y_r(k+d)-f[y(k),y(k-1),\ldots,y(k-n+1),u(k-1),\ldots,u(k-n+1)]}{g[y(k),y(k-1),\ldots,y(k-n+1),u(k-1),\ldots,u(k-n+1)]}$$

其中，$d\geqslant 2$。图 15-23 显示了 NARMA-L2 控制器的结构。

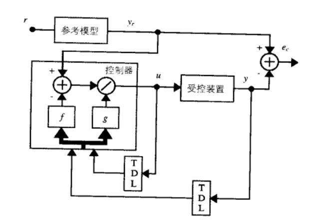


图 15-23 NARMA-L2 神经网络控制器的结构

此控制器可以用在前面提到的 NARMA-L2 系统辨识模型上，如图 15-24 所示。

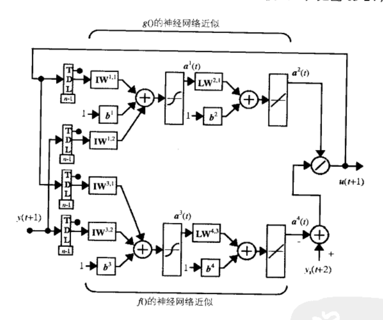


图 15-24 NARMA-L2 神经网络控制器的结构

15.3.3 NARMA-L2 控制器 Simulink 实例

本节将显示如何训练一个 NARMA-L2 控制器，实现对系统辨识和控制。此模型是基于 Simulink 的，如下例所示。

【例 15-2】 NARMA-L2 控制器实例。对于图 15-25 所示的系统，有一块磁铁，受到垂直方向的运动约束，其下方有一块电磁铁，当电磁铁通电之后，就会对上方的磁铁产生

电磁力的作用。试对此磁铁的位置进行控制，使得磁铁能够悬浮在空中而不掉下来。

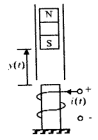


图 15-25　悬浮磁铁系统示意图

此系统运动的动力学方程为：

$$\frac{\mathrm{d}^2 y(t)}{\mathrm{d}t^2}=-g+\frac{\alpha}{M}\cdot\frac{i^2(t)}{y(t)}-\frac{\beta}{M}\cdot\frac{\mathrm{d}y(t)}{\mathrm{d}t}$$

其中，$y(t)$ 为磁铁距离电磁铁的高度，$i(t)$ 为电磁铁中的电流强度，M 为磁铁质量，g 为重力加速度。参数 β 为粘滞摩擦系数，由磁铁所在容器的材料决定，α 为场强常数，由电磁铁线圈匝数和磁铁磁性大小决定。

解： ① 建立模型。在 MATLAB 命令空间中输入命令：

```
narmamaglev
```

此命令将直接启动 Simulink 界面，同时弹出如图 15-26 所示窗口，其中包含了此磁铁控制系统模型。

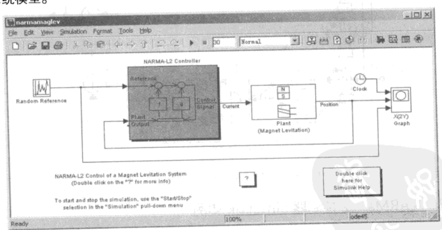


图 15-26　悬浮磁铁控制系统 Simulink 模型

在此窗口中，包含磁铁悬浮模块（Magnet Levitation）、NARMA-L2 控制器模块，信号源模块（Random Reference）、显示模块（Graph）。双击 Magnet Levitation 模块可以查看其内部结构，如图 15-27 所示。

图 15-26 中的 NARMA-L2 控制器模块是在神经网络工具箱中生成和复制过来的，此模块的输出信号与悬浮磁铁模块连接，作为控制信号；而磁铁悬浮模块的输出信号又反馈到 NARMA-L2 控制器模块的输入端。

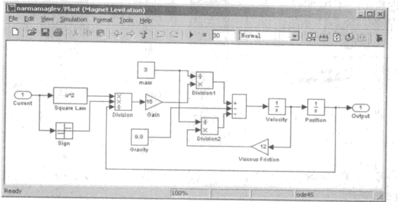


图 15-27 悬浮磁铁模块内部结构

② 系统辨识。

双击 NARMA-L2 Controller 模块图标，将弹出如图 15-28 所示的新窗口。

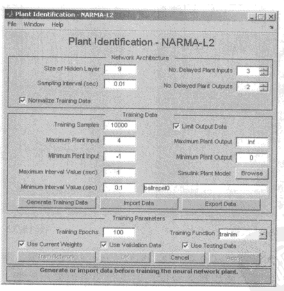


图 15-28 NARMA-L2 控制器窗口

此窗口与 15.2.3 节中介绍的“Plant Identification”窗口类似，只需要单击“Generate Training Data”按钮，然后单击“Accept Data”按钮，之后再单击“Train Network”按钮即可开始对网络系统模型的训练。其中产生的训练数据如图 15-29 所示。

经过训练后，输出的结果如图 15-30、15-31、15-32 所示。

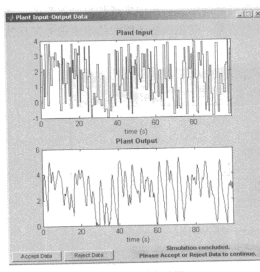


图 15-29 训练样本数据

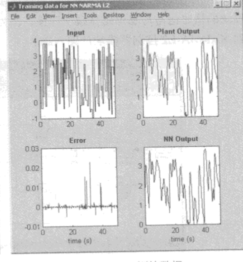




图 15-30 训练数据

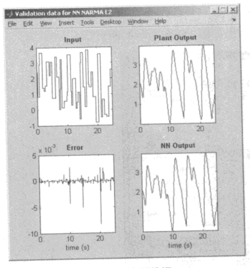


图 15-31 验证数据

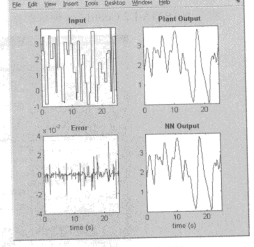


图 15-32 测试数据

其中，图 15-30 为训练数据结果，图 15-31 为验证数据结果，图 15-32 为测试数据结果。这几张图中，左上角为随机输入数据，右上角为系统输出数据，左下角为误差曲线，右下角则为网络输出数据。对网络进行训练时将弹出如图 15-33 所示的 nntraintool 窗口。

③ 系统仿真。

单击“Plant Identification”窗口中的“OK”按钮，将会在 NARMA-L2 Controller 模块中载入训练过后的神经网络系统模型。

回到 Simulink 界面下，单击“Simulation”菜单下的“Start”选项，将会得到如图 15-34 的结果。

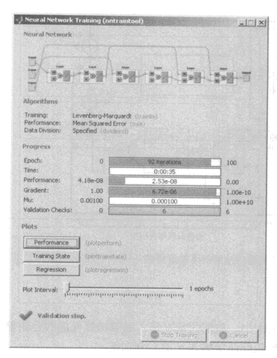


图 15-33 nntraintool 窗口

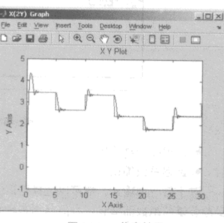


图 15-34 仿真结果

15.4 神经网络模型参考控制

神经网络模型参考控制系统是由一个控制器网络和一个系统模型网络构成的，其结构如图 15-35 所示。

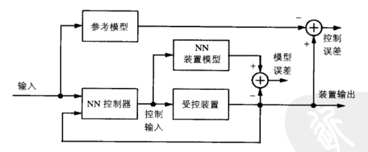


图 15-35 神经网络模型参考控制原理

在控制时，首先进行受控装置的系统辨识，然后训练控制器，使得系统输出能够跟踪参考模型的输出。图 15-36 为图 15-35 中神经网络控制器与神经网络装置模型的结构示意图。其中，神经网络控制器的输入共有三种类型，分别是：

- 延迟参考输入；

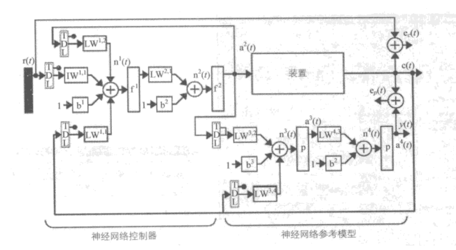


图 15-36　神经网络控制器与神经网络装置模型

- 延迟控制器输出；
- 延迟系统输出。

对于每一种输入，都可以选择延迟值的大小。典型的情况是，隐层神经元的数目随着受控装置的维数而增长。神经网络系统模型包含两种输入类型，分别是：

- 延迟控制器输出；
- 延迟系统输出。

同控制器一样，也可以设置延迟的数目，后面将演示如何对这些参数进行设置。

下面介绍如何对神经网络控制器进行训练，实现对系统的控制。此模型实例是基于 Simulink 程序模块的，如下例所示。

【例 15-3】　模型参考控制实例。图 15-37 所示的系统为一个由直流电机驱动的单臂机器人，圆轮代表直流电机驱动转轴，斜线代表机器臂。系统的目标是控制此单臂机器人的运动。

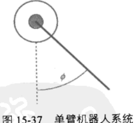


图 15-37　单臂机器人系统

此系统运动的动力学方程为：

$$\frac{d^2\phi}{dt^2}=-10\sin\phi-2\frac{d\phi}{dt}+u$$

其中，参数 ϕ 为机器臂相对竖直方向的角度，u 为直流电动机提供的转动力矩。希望对控制器进行训练，从而使得机器臂的运动跟踪参考模型如下：

$$\frac{d^2y_r}{dt^2}=-9y_r-6\frac{dy_r}{dt}+9r$$

其中，y_r 是参考模型的输出，r 是输入参考信号。

解: 应用一个 5-13-1 结构的神经网络控制器实现目标功能，控制器包含两个延迟参考输入、两个延迟装置输出，以及一个延迟控制器输出。采样率设定为 0.05s。

① 建立模型。在 MATLAB 命令空间中输入命令：

```
mrefrobotarm
```

此命令将直接启动 Simulink 界面，同时弹出如图 15-38 所示窗口，其中包含了神经网络模型参考控制模块。

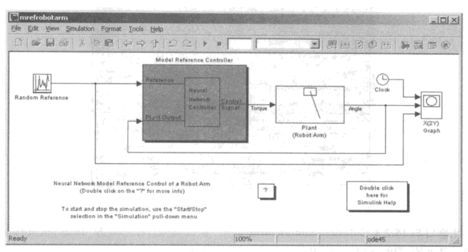


图 15-38 模型参考控制器 Simulink 实例

此窗口包含：单臂机器人模块（Robot Arm）、模型参考控制器模块、参考信号源模块（Random Reference）和显示模块（Graph）。双击“Robot Arm”模块图标可以查看其内部结构，如图 15-39 所示。

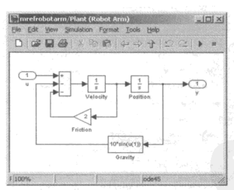


图 15-39 Robot Arm 模块结构

图 15-38 中的模型参考控制器模块是在神经网络工具箱中生成和复制过来的，此模块的输出信号与单臂机器人模块连接，作为控制信号；而单臂机器人模块的输出信号又反馈到控制器模块的输入端。

② 系统辨识。

双击“Model Reference Controller”模块图标，将弹出如图 15-40 所示的新窗口。

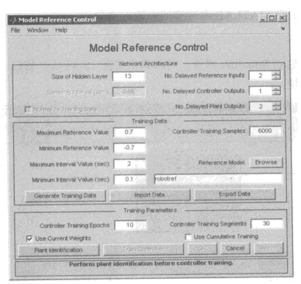


图 15-40 模型参考控制窗口

此窗口与 15.2.3 节中的“Plant Identification”窗口类似。首先单击“Plant Identification”按钮，将弹出“Plant Identification”窗口，如图 15-41 所示。

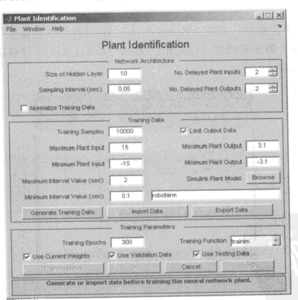


图 15-41 “Plant Identification”窗口

在“Plant Identification”窗口中单击“Generate Training Data”按钮，产生的训练数据如图 15-42 所示。

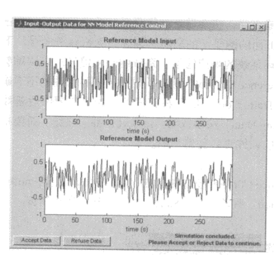


图 15-42 训练数据

然后单击“Accept Data”按钮，回到“Plant Identification”窗口，单击“Train Network”按钮对网络模型进行训练，会显示如图 15-43 和图 15-44 所示的窗口。

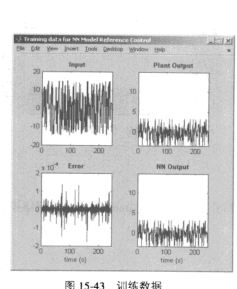


图 15-43 训练数据

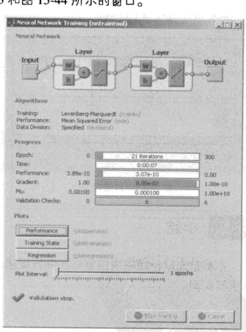


图 15-44 nntraintool 窗口

单击“Plant Identification”窗口中的“OK”按钮，回到“Model Reference Control”窗口，之后再单击“Train Controller”按钮即可开始对网络控制器的训练。

程序将数据分段输入网络，并且对网络进行指定次数的迭代，直到所有训练数据都输入网络为止。控制器训练所需的时间要比系统模型训练长很多，这是因为它采用了动态反

向传播算法。训练完成之后，将会显示此闭环系统的响应结果，如图 15-45 所示。

在图 15-45 中，上面的图显示的是用于训练的随机参考信号，下面的图显示的是参考模型的响应信号和闭环系统的响应信号。系统的响应信号应该跟踪参考模型的输入信号。

返回“Model Reference Control”窗口，如果控制器的性能不准确，则需要再次对控制器进行训练，单击“Train Controller”按钮即可，如果需要使用新数据对其进行训练，则单击“Generate Training Data”按钮或者“Import Data”按钮。需要指出的是，如果系统模型不准确，则控制器的训练也会受到影响。

③ 系统仿真。

在“Model Reference Control”窗口中单击“OK”按钮，回到 Simulink 下的“mrefrobotarm”窗口，选择“Simulation”菜单中的“Start”选项，开始系统仿真。绘出的系统输出与参考信号如图 15-46 所示。

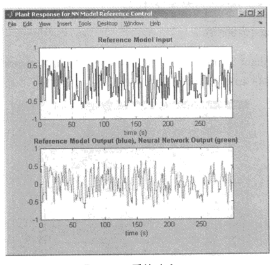


图 15-45 系统响应

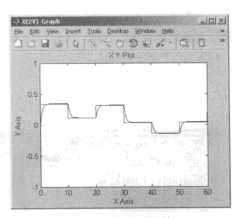


图 15-46 仿真输出结果

15.5 小结

本章首先对神经网络控制系统进行概述，然后介绍 3 个基于 Simulink 的神经网络控制系统的实例。

第 16 章　神经网络故障诊断

故障诊断是近 40 年发展起来的新学科，它是适应工程需要而发展起来的多学科交叉的综合学科。工业故障诊断通常包含两种含义，一种是指利用某些专用的仪器检测工业设备的运转是否正常；另一种是指由计算机完成工况分析，对故障模式、故障原因、故障程度等问题进行分析、判断，得出结论。

在各种基于信号处理的故障诊断算法之中，神经网络为故障诊断提供了一种新的解决途径。特别是对于实际中难以建立数学模型的复杂系统，神经网络的优势更为明显。本章将对神经网络在不同工业领域内的故障诊断应用进行实例说明和介绍，主要内容包括如下几部分。

（1）神经网络故障诊断概述。

（2）基于神经网络的滚动轴承故障诊断。

（3）基于神经网络的汽车 ABS 防抱死系统故障诊断。

（4）基于神经网络的柴油机故障诊断。

（5）基于神经网络的水循环系统故障诊断。

16.1 神经网络故障诊断概述

现代化设备日趋大型化、复杂化、自动化和连续化，设备的故障诊断技术越来越受到重视。如果一台机械设备在出现故障征兆时没有被及时发现，那么故障发生后，轻则停工停产，经济受到损失，重则发生重大事故，后果不堪设想。为此，研制和开发出自动故障诊断系统，在出现故障征兆时能够给出预警信号，及时排除故障隐患，对于现代企业来说具有重大的经济意义。

机械故障诊断的根本目标是确诊连续运行机器的潜在故障，保证机器安全有效的运行。所谓确诊就是能够准确地诊断出机器的故障类型、故障严重程度及故障的具体位置。从本质上讲，故障诊断就是模式识别的问题。

自然界的事物和现象通常可分为多个相似但又不完全相同的事物或个体组成的类别，人们把这样的类别称为模式类或模式，而把其中每一个事物或现象称为该模式的一个样本。同类的样本彼此相似，具有某些共同的特征，不同类的样本彼此互不相似。所谓的模式识别就是从模式空间到类别隶属空间的正确映射。

在模式识别映射过程中，需要用数学语言来描述。所以在模式识别的学习及识别之前，先要对待识别的目标进行抽象，即对待研究对象重要的特征或属性进行测量并将结果数字化，或将对象分解并符号化，形成特征矢量或符号串、关系图，从而产生代表对象的模式。

故障诊断的关键是实现从故障征兆空间到故障空间的映射，从而实现对故障的识别和诊断（模式识别）。传统的方法是采用基于符号推理的专家系统，但专家系统用于故障诊断存在知识获取困难、组合爆炸和匹配冲突等难以克服的问题，使其应用达不到预期的效果。

神经网络具有自学习能力、非线性映射能力、对任意函数的逼近能力、并行计算能力和容错能力，这些能力为构造新型故障诊断系统提供了有力保障。就运行过程而言，机械设备实质上是一个复杂的非线性动力系统，特别是在多故障和非平稳状态下，要求模式识别的过程具有自适应性和鲁棒性，也就是要求模式分类器具有自适应地处理由噪声引起的模式失真的能力，能够根据设备运行参数的变化调整分类过程，能够根据输入模式的数据对存储器和分类器的结构进行自适应调节。基于神经网络的故障诊断系统，就是用故障征兆的可信度作为输入，经过神经网络的并行数值计算，输出对应故障的可信度，从而完成故障模式的识别，因而机械故障诊断领域是神经网络的重要应用领域之一。

神经网络应用于故障诊断的步骤通常如下。

① 通过信号监测和分析，抽取反映被检测对象（设备、部件或零件等）的特征参数，如（$x1$，$x2$，…，xn）作为网络的输入模式。

② 对被检测对象的状态类别进行编码。例如，对于正常、故障 1、故障 2 三种状态，可将期望输出编码为：正常（0，0）、故障 1（0，1）、故障 2（1，0）。

③ 进行网络设计，确定网络层数和各层神经元数。输入层单元数由特征参数个数决定；输出层单元数由状态数和状态编码方式确定；隐层一般为 1 层，问题复杂时可取 2 层，隐层单元数的选择原则目前尚无理论依据，可根据问题规模大小凭经验确定。

④ 用各种状态样本组成训练样本，输入网络，对网络进行训练，确定各单元的连接权值。

⑤ 用训练好的网络对待检对象进行状态识别，即把待检对象的特征参数作为网络输入，根据网络输出确定待检对象的状态类别。

为提高网络的故障诊断性能，可把使用中发现的错误判断作为训练样本加入训练样本集，对网络进行进一步训练，从而使网络的性能得到改善。

以下各节介绍神经网络故障诊断的应用实例。

16.2 基于神经网络的滚动轴承故障诊断

16.2.1 问题背景

滚动轴承是由内环、外环、滚动体和保持架 4 种元件组成。通常，其内环与机械传动轴的轴颈过盈配合连接，工作时与轴一起转动；外环安装在轴承座、箱体或者其他支撑物上，工作时一般外环固定，但也有外环回转、内环不动或内外环分别按不同的转速回转的使用情况。

滚动体是滚动轴承的核心元件，它使相对运动表面间的滑动摩擦变成滚动摩擦。滚动

体的形式有球形、圆柱形、圆锥形、鼓形等。在滚动轴承内外环上都有凹槽滚道，它们起着降低接触应力和限制滚动轴承轴向移动的作用。保持架使滚动体等距离分布并减少滚动体间的摩擦和磨损。如果没有保持架，相邻滚动体将直接接触，且相对摩擦速度是表面速度的两倍，发热和磨损都较大。

滚动轴承（球轴承）典型结构示意如图 16-1 所示。

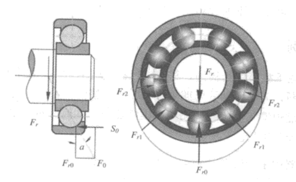


图 16-1 典型滚珠轴承结构

滚动轴承在工作过程中，由于装配不当、润滑不良、水分和异物侵入、腐蚀和过载等都可能使轴承损坏。滚动轴承主要损伤形式包括：

- 磨损失效；
- 疲劳失效；
- 腐蚀失效；
- 断裂失效；
- 压痕失效；
- 胶合失效；
- 保持架损坏。

滚动轴承故障诊断的目的是保证轴承在一定的工作环境中承受一定荷载以一定的转速运转、在一定的工作期间内可靠有效地运行，以保证整个机器的工作精度。与此目的相对应，轴承故障诊断就要通过对能够反映轴承工作状态的信号进行观测、分析和处理来识别轴承的状态。所以，从一定程度上说，轴承故障诊断就是轴承的状态识别。

完整的轴承故障诊断过程包括以下 5 个方面的内容。

（1）信号测取。根据轴承的工作环境和性质，选择并测量能够反映轴承工况或状态的信号。

（2）特征提取。以一定的信号分析与处理方法从测量的信号中抽取出能够反映轴承状态的有用信息。

（3）状态识别。以一定的状态识别方法识别轴承的状态，即简单判断轴承工作是否有故障。

（4）状态分析。根据征兆，进一步分析有关状态的情况以及发展趋势。当有故障时，

详细分析故障类型、性质、部位、产生原因与趋势等。

（5）决策干预。根据轴承状态及其发展趋势，做出决策，如调整、控制，或继续监视等。

轴承故障诊断的目的是从故障定位到确定故障性质，进而确定故障发生的程度。由于神经网络具有处理复杂多模式的能力，以及进行联想、推测和记忆的功能，因而适于应用在滚珠轴承的故障诊断上。下面介绍一个故障诊断实例。

16.2.2 问题实例

【例 16-1】 轴承故障诊断实例。利用 BP 神经网络对石油钻井的绞车及传动机组滚动轴承进行故障诊断，能够在轴承早期故障时发出预警信号，提前对将要发生故障的轴承进行维修或更换，缩短停工停产时间和减小维修费用，从而使石油生产损失减少到最低，保证石油生产顺利安全进行。

选取某型减速器的主动轴滚动轴承的 4 个特征参数，包括均方根值、峭度、谐波指标和 SQ 参数，这 4 个参数组成输入样本向量，实测数据如表 16-1 所示。

表 16-1 试验轴承特征参数

轴承状态	均方根值	峭　度	谐波指标	SQ 参数
正常新轴承	0.64	1.37	0.71	0.78
正常新轴承	0.68	1.31	0.64	1.31
正常新轴承	0.91	1.35	0.75	1.59
正常新轴承	0.69	1.38	0.68	0.9
内圈点蚀	8.24	2.23	0.99	2
滚珠点蚀	2.01	1.65	0.94	4.39
保持架损坏	0.93	1.33	0.73	1.54
外圈严重裂纹	3.89	2.01	0.88	20.1
外圈较轻裂纹	1.65	1.66	0.9	4.48
外圈微裂纹	1.35	1.39	0.95	2.89

解： ① 定义输入样本数据。从表 16-1 的 10 组数据中选择 7 组作为输入样本，在 MATLAB 命令空间中输入命令：

```
P11=[0.64   1.37   0.71   0.78]';
P12=[0.68   1.31   0.64   1.31]';
P21=[1.65   1.66   0.9    4.48]';
P22=[1.35   1.39   0.95   2.89]';
P31=[8.24   2.23   0.99   2]';
P41=[2.01   1.65   0.94   4.39]';
P51=[0.93   1.33   0.73   1.54]';
P=[P11 P12 P21 P22 P31 P41 P51];
```

② 对输出状态进行编码，输出为四维向量，定义期望输出向量如表 16-2 所示。

表 16-2 轴承状态编码对照表

类 别 号	轴承状态	输出向量
1	正常轴承	(0000)
2	外圈裂纹	(1000)
3	内圈点蚀	(0100)
4	滚珠点蚀	(0010)
5	保持架损坏	(0001)

定义输出向量。输入命令：

```
t11=[0 0 0 0]';
t12=[0 0 0 0]';
t21=[1 0 0 0]';
t22=[1 0 0 0]';
t31=[0 1 0 0]';
t41=[0 0 1 0]';
t51=[0 0 0 1]';
t=[t11 t12 t13 t21 t22 t31 t41 t51];
```

③ 构建 BP 神经网络，网络参数如表 16-3 所示。

表 16-3 BP 神经网络参数设定

网络层数	隐层神经元数目	传递函数	训练算法
3 层	[8, 4]	{'logsig','purelin'}	'trainlm'

接下来应用输入和输出样本向量对网络进行训练。输入命令：

```
net=newff(minmax(P),[8,4],{'logsig','purelin'},'trainlm'),
net.trainParam.show = 100,
net.trainParam.epoch = 2000,
net.trainParam.goal= 1e-3,
[net,tr]=train(net,P,t),
```

训练过程中，网络误差的变化情形如图 16-2 所示。

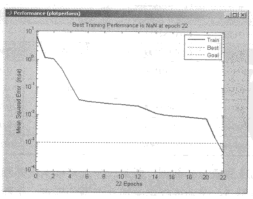


图 16-2 网络训练过程中的误差变化曲线

④ 利用所有 10 个样本对经过训练的神经网络进行测试检验。输入如下命令：

```
Ptest11=[0.64   1.37   0.71   0.78]';
Ptest12=[0.68   1.31   0.64   1.31]';
Ptest13=[0.91   1.35   0.75   1.59]';
Ptest14=[0.69   1.38   0.68   0.9]';
Ptest21=[3.89   2.01   0.88   20.1]';
Ptest22=[1.65   1.66   0.9    4.48]';
Ptest23=[1.35   1.39   0.95   2.89]';
Ptest31=[8.24   2.23   0.99   2]';
Ptest41=[2.01   1.65   0.94   4.39]';
Ptest51=[0.93   1.33   0.73   1.54]';
Ptest=[Ptest11 Ptest12 Ptest13 Ptest14 Ptest21 Ptest22 Ptest23 Ptest31
Ptest41 Ptest51];
result_test = sim(net, Ptest)'
```

输出结果为：

```
result_test =
    0.0002    0.0009   -0.0003   -0.0051
   -0.0010   -0.0016    0.0002   -0.0047
    0.0518    0.0268   -0.0104    1.0687
    0.0318    0.0183   -0.0072   -0.3239
    0.5137   -0.1727    0.0008    1.7685
    0.9999   -0.0019    0.0017   -0.0042
    0.9998    0.0011   -0.0005    0.0017
    0.0003    0.9979    0.0005   -0.0046
    0.0005   -0.0018    1.0011   -0.0035
    0.0006    0.0001   -0.0000    0.9939
```

绘成表格如表 16-4 所示。

表 16-4　诊断结果

实际轴承状态	诊断数据				诊断状态
类别 1	0.000169	0.0008883	–0.000261	–0.00509	类别 1
类别 1	–0.00097	–0.001555	0.0001634	–0.00472	类别 1
类别 1	0.051824	0.0267997	–0.010418	1.068747	类别 5
类别 1	0.03176	0.0182837	–0.007157	–0.32388	类别 1
类别 2	0.513654	–0.172699	0.0007695	1.76854	类别 5
类别 2	0.99994	–0.001884	0.0017347	–0.00423	类别 2
类别 2	0.999806	0.0011026	–0.000465	0.001668	类别 2
类别 3	0.000274	0.9978975	0.0005375	–0.00459	类别 3
类别 4	0.000489	–0.001779	1.0010996	–0.00348	类别 4
类别 5	0.000591	0.0001066	–2.03E-05	0.993899	类别 5

可以看到，表中加灰底表示的数据，即第三个正常样本、外圈严重裂纹样本的诊断出现了错误，均误判为类别 5。在训练样本中加入这两个样本，重新对网络进行训练后再诊断，同时改变网络参数，设置中间层为 10 个神经元，则可以纠正这两处错误。

再次进行训练，网络的误差曲线如图 16-3 所示。

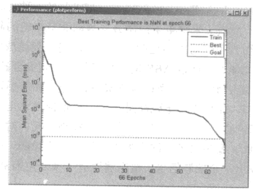


图 16-3　再次训练过程中误差变化曲线

训练经过 66 次迭代之后达到了期望误差限，对样本重新诊断的结果如表 16-5 所示。

表 16-5　再次诊断的结果

实际轴承状态	诊断数据				诊断状态
类别 1	0.00507	0.0017823	0.005497	−0.00384	类别 1
类别 1	−0.00309	0.0005237	−0.004101	0.008815	类别 1
类别 1	0.008517	0.0007454	0.003831	0.057907	类别 1
类别 1	−0.07063	0.106827	−0.061613	−0.00869	类别 1
类别 2	1.027242	1.03E-05	−0.007776	0.012847	类别 2
类别 2	0.999035	0.0008227	0.0031204	−0.01018	类别 2
类别 2	0.997089	0.0024052	0.0020056	−0.00587	类别 2
类别 3	0.020507	0.9818684	0.0068985	0.052362	类别 3
类别 4	0.004922	0.0024127	0.9990207	−0.0002	类别 4
类别 5	−0.00903	−0.002148	−2.95E-03	0.940167	类别 5

可以看到，经重新训练改进后，神经网络对各个样本的诊断结果均正确。

16.3 基于神经网络的汽车防抱死系统故障诊断

16.3.1 问题背景

汽车故障诊断是依靠先进的传感器技术和检测技术，采集汽车各个系统的各种动态信息，并对这些信息进行分析、处理、区分、识别，确认其是否属异常表现，预测其发展趋势，查明其产生原因、发生部位和严重程度，提出针对性的维修措施和处理方法。

车轮抱死将导致汽车可能会出现下面三种情况。

（1）制动距离变长。

（2）方向稳定性变差，出现侧滑现象，严重时出现旋转掉头。

（3）方向操纵性丧失，驾驶员不能控制汽车的行驶方向。

防抱死制动系统（Anti-lock Braking System，ABS）是一种主动安全装置，它在制动过程中根据“车辆-路面”状况，采用电子控制方式自动调节车轮的制动力矩来达到防止车轮抱死的目的。ABS 的引入不仅可以防止制动过程中后轮抱死造成的侧滑甩尾，同时可以防止前轮抱死而丧失转向能力。所以 ABS 是一种有效的车辆安全装置。

对于 ABS 的故障诊断，我们可以通过对 ABS 的执行器和传感器的诊断来减少或避免 ABS 故障的发生。在汽车 ABS 中，电磁阀是一个关键性部件，当电磁阀发生故障时，应该关闭 ABS，进行常规制动。ABS 中，ECU 为电子控制单元，对电磁阀进行控制，控制信号即是输出信号。因此，要想对各电磁阀的工作情况进行诊断，一般要增设故障诊断电路，即 ECU 向执行器发出一个控制信号，执行器要有一条专用回路来向 ECU 反馈其执行情况。

不同公司开发的 ABS，其电子控制单元和控制逻辑一般不同，但电磁阀的结构形式和工作原理却基本类似。图 16-3 所示为某重型商用车 ABS 的基本组成示意图。

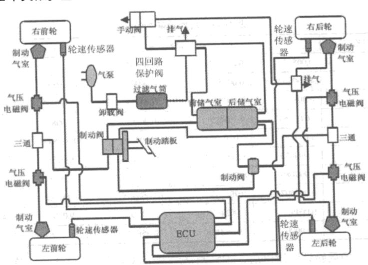


图 16-3　某重型商用车 ABS 示意图

由图 16-3 中可以看出，制动压力调节器，即电磁阀，是 ABS 中最主要的执行机构，它串联在气制动管路中。每个控制通道中各设置一个制动压力调节器（即制动气室），每个制动压力调节器分别对相应车轮进行控制。它接收来自 ECU 的电压信号，便可实施增压、减压、保压等动作，以控制制动力，达到制动过程中车轮既不抱死又能较好地利用路面峰值附着系数的目的。调节器的结构原理及性能特征对 ABS 的控制品质具有非常重要的影响。

如果汽车 ABS 执行器与传感器发生故障，驾驶员踏下脚踏板时，气体压力将直接进入制动气室，车轮突然抱死。

BP 神经网络由于具有较强的非线性映射能力而被广泛应用于故障诊断领域。它通过

对故障实例的训练和学习，用分布在神经网络中的连接权值来表达所学习的故障诊断知识，具有对故障的联想记忆、模式匹配和相似归纳的能力，可以实现故障与征兆之间复杂的非线性映射关系，因此，此处采用 BP 神经网络对汽车 ABS 执行器与传感器发生故障时的行为模式进行识别与诊断。

16.3.2 问题实例

【例 16-2】 ABS 防抱死系统故障诊断实例。利用 BP 神经网络对某型卡车的 ABS 故障模式进行诊断。

首先对调节器故障类型进行分类，故障原因分为 5 类，以 $y1$、$y2$、$y3$、$y4$、$y5$ 分别代表无故障、左前调节器故障、右前调节器故障、左后调节器故障和右后调节器故障。以故障模式 $X=(x1, x2, x3, x4, x5, x6)$ 作为输入，各故障模式 x 分量分别代表纵向车速、侧向车速以及四轮轮速的相应变化。以故障模式各分量的采集数据作为 BP 神经网络网络输入，以故障原因作为神经网络的输出，进行故障模式诊断。

取 50 个样本，分别为 0.3s、0.4s、0.5s、0.6s、1.5s、1.8s、3.2s、3.8s、4.2s、4.5s 时的速度值，采集的样本具体如表 16-6 所示（表中各速度均为相对数值）。

表 16-6 调节器（ABS 阀）故障时采集的速度数据

故障类型	纵向车速	横向车速	左前轮速	右前轮速	左后轮速	右后轮速
无故障（0.3s）	22.082	0	21.069	21.069	21.308	21.308
无故障（0.4s）	21.813	0	20.673	20.673	20.884	20.884
无故障（0.5s）	21.509	0	20.299	20.299	20.47	20.47
无故障（0.6s）	21.179	0	19.928	19.928	20.072	20.072
无故障（1.5s）	17.781	0	16.615	16.615	16.684	16.684
无故障（1.8s）	16.602	0	15.51	15.51	15.572	15.572
无故障（3.2s）	11.077	0	10.347	10.347	10.388	10.388
无故障（3.8s）	8.7086	0	8.1346	8.1346	8.1668	8.1668
无故障（4.2s）	7.1293	0	6.6594	6.6594	6.6857	6.6857
无故障（4.5s）	5.9448	0	5.553	5.553	5.575	5.575
左前故障（0.3s）	21.813	0.39464	13.601	20.947	20.553	21.104
左前故障（0.4s）	21.484	0.78104	0	20.504	19.87	20.702
左前故障（0.5s）	21.14	1.199	0	20.099	19.191	20.294
左前故障（0.6s）	20.77	1.6298	0	19.692	18.375	19.879
左前故障（1.5s）	16.683	5.3032	0	15.667	0	15.747
左前故障（1.8s）	15.134	6.3275	0	14.188	0	14.229
左前故障（3.2s）	6.1988	10.297	0	5.6033	0	5.52
左前故障（3.8s）	0.62346	11.198	0	0	0	0
左前故障（4.2s）	0	11.207	0	0	0	0
左前故障（4.5s）	0	11.207	0	0	0	0

续表

故障类型	纵向车速	横向车速	左前轮速	右前轮速	左后轮速	右后轮速
右前故障（0.3s）	21.813	−0.3946	20.947	13.601	21.104	20.553
右前故障（0.4s）	21.484	−0.781	20.504	0	20.702	19.87
右前故障（0.5s）	21.14	−1.199	20.099	0	20.294	19.191
右前故障（0.6s）	20.77	−1.6298	19.692	0	19.879	18.375
右前故障（1.5s）	16.683	−5.3032	15.667	0	15.747	0
右前故障（1.8s）	15.134	−6.3275	14.188	0	14.229	0
右前故障（3.2s）	6.1988	−10.297	5.6033	0	5.52	0
右前故障（3.8s）	0.62334	−11.198	0	0	0	0
右前故障（4.2s）	0	−11.207	0	0	0	0
右前故障（4.5s）	0	−11.207	0	0	0	0
左后故障（0.3s）	21.953	0.18066	20.891	21.039	18.766	21.217
左后故障（0.4s）	21.619	0.39155	20.362	20.607	12.813	20.793
左后故障（0.5s）	21.279	0.65987	19.884	20.196	0	20.398
左后故障（0.6s）	20.927	0.94638	19.472	19.804	0	19.998
左后故障（1.5s）	17.236	3.4358	15.966	16.176	0	16.27
左后故障（1.8s）	15.894	4.1516	14.784	14.899	0	14.972
左后故障（3.2s）	9.1394	6.8422	8.6274	8.4861	0	8.4864
左后故障（3.8s）	5.961	7.655	5.7093	5.4665	0	5.4482
左后故障（4.2s）	3.723	8.0448	3.6475	3.3386	0	3.3163
左后故障（4.5s）	1.9715	8.2341	2.0296	1.6719	0	1.6552
右后故障（0.3s）	21.953	−0.1806	21.039	20.891	21.217	18.766
右后故障（0.4s）	21.619	−0.3915	20.607	20.362	20.193	12.873
右后故障（0.5s）	21.279	−0.6598	20.196	19.884	20.398	0
右后故障（0.6s）	20.927	−0.9463	19.804	19.472	19.998	0
右后故障（1.5s）	17.236	−3.4358	16.176	15.966	16.27	0
右后故障（1.8s）	15.894	−4.1516	14.899	14.784	14.972	0
右后故障（3.2s）	9.1394	−6.8422	8.4861	8.6274	8.4864	0
右后故障（3.8s）	5.961	−7.655	5.4665	5.7093	5.4482	0
右后故障（4.2s）	3.723	−8.0448	3.3386	3.6475	3.3163	0
右后故障（4.5s）	1.9715	−8.2341	1.6719	2.0296	1.6552	0

解： ① 定义输入样本数据，从表 16-6 的 50 组数据中选择 45 组作为输入样本，此处取 0.3s、0.4s、0.5s、0.6s、1.8s、3.2s、3.8s、4.2s、4.5s 时的 9 组速度值样本作为训练样本，1.5s 时的速度值作为检测样本，将输入样本存储于 Excel 文件 16_2_P.xls 中，利用 MATLAB 命令读取。

在 MATLAB 命令空间中输入命令：

```
[filename, pathname] = uigetfile('11_2_P.xls');
file=[pathname filename];
```

```
x=xlsread(file);
p=x;
p=p';
```

p 就是定义的输入样本向量。

② 首先对故障类型进行编码，编码如表 16-7 所示。

表 16-7 调节器输出向量编码表

编 号	故障类型	输出向量
1	无故障	(10000)
2	左前故障	(01000)
3	右前故障	(00100)
4	左后故障	(00010)
5	右后故障	(00001)

输出向量为 5 维向量，其中故障类型位为 1 则代表发生了相应的故障。接下来定义期望输出响应向量。输入命令：

```
T=zeros(45,5);
for j=0:4
    for i=j*9+1:j*9+9
        T(i,j+1)=1;
    end
end
T=T',
```

T 为定义的期望响应向量。

③ 构建 BP 神经网络，网络参数如表 16-8 所示。

表 16-8 BP 神经网络参数

网络层数	各层神经元数目	传递函数	训练算法
3 层	[12, 5]	{'logsig','purelin'}	'trainlm'

接下来对网络进行训练。输入命令：

```
net=newff(minmax(p),[12,5],{'logsig','purelin'},'trainlm')
net.trainParam.show = 100,
net.trainParam.epoch = 2000,
net.trainParam.goal= 1e-3,
[net,tr]=train(net,p,T),
```

执行上面的命令，弹出 nntraintool 窗口，单击窗口中的“Performance”按钮，绘出曲线如图 16-4 所示。

可以看到，经过 13 次迭代，网络达到了设定的误差性能目标。

④ 将 1.5s 时的数据存入 Excel 文件 16_2_Ptest.xls，利用 xlsread 命令读取输入检验向量，对网络诊断效果进行测试。输入命令：

```
[filename, pathname] = uigetfile('16_2_Ptest.xls');
```

```
file=[pathname filename];
xtest=xlsread(file);
ptest=xtest';
result_test = sim(net, ptest)',
```

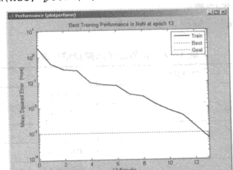


图 16-4　训练过程中的误差变化曲线

仿真输出结果为：

```
result_test =
    0.9650   -0.0093    0.0507   -0.0257   -0.0002
    0.0675    1.0615   -0.1104    0.0983   -0.1219
   -0.0246   -0.0324    0.9633    0.0564    0.0539
    0.0782    0.0256   -0.0828    1.0163   -0.0356
   -0.0141   -0.0353    0.0154   -0.0153    1.0427
```

将结果整理为表格，即如表 16-9 所示。

表 16-9　诊断结果

实际故障状态	诊断数据					诊断状态
无故障（1.5s）	0.97	–0.01	0.05	–0.03	0.00	无故障
左前故障（1.5s）	0.07	1.06	–0.11	0.10	–0.12	左前故障
右前故障（1.5s）	–0.02	–0.03	0.96	0.06	0.05	右前故障
左后故障（1.5s）	0.08	0.03	–0.08	1.02	–0.04	左后故障
右后故障（1.5s）	–0.01	–0.04	0.02	–0.02	1.04	右后故障

可见，训练后的网络对 ABS 故障模式的诊断结果完全正确。

16.4 基于神经网络的柴油机故障诊断

16.4.1 问题背景

柴油机作为大部分农业机械的动力设备，其运行状况的好坏直接影响着农民的经济效益。另外，柴油机也可作为车辆、船舶、电站等常用设备的动力装置，其基本结构如图 16-5 所示。因此，对柴油机进行状态检测和故障诊断，及时准确地排除故障，确保设备安全运

行十分重要。

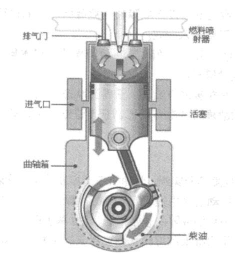


图 16-5 基本二冲程柴油机结构

柴油机的主要故障类型可分为两大类：一类是机械式故障，如各种间隙因磨损而增大，引起振动的幅频变化；另一类是燃烧系统的热力学故障，如喷油角变化，进排气机构故障等。基于柴油机振动信号的神经网络可以对以上各种故障模式进行诊断以及甄别。现将柴油机主要的故障形式列举如下。

（1）气缸-活塞故障。

（2）曲柄-连杆机构故障。

（3）活塞环漏气。

（4）喷油角变化。

（5）燃油雾化质量不良。

（6）气门机构故障。

（7）气门漏气。

基于神经网络的诊断方法就是利用神经网络对柴油机故障进行模式分类。神经网络直接用于故障诊断时，要挑选特征参数组成输入向量，以故障原因作为输出向量，利用典型样本学习所得权值进行模式识别。

对于故障信号的选择，可以测量振动信号，也可以测量气体最高爆发压力、油压波形圈、高压油管的振动信号，两种方法各有各的特点。相对来说，振动信号的测量更简单易行。

另一方面，柴油机工作时产生的振动信号包含着极为丰富的信息，是反映系统状态及其变化规律的主要信号，利用振动信号对故障进行诊断，是柴油机故障诊断中最有效、最常用的方法。需要指出的是，故障征兆集的设计，即故障信息特征量的提取对故障诊断至关重要。特征量过多，势必造成网络训练、决策计算量过大，不易得到正确的结果；但是特征量过少，判据不足，诊断效果也不好。

基于神经网络的柴油机振动信号故障诊断分 3 步实现。

① 通过试验获得给定工况在设定故障和无故障状态下的过程参数，经预处理提取故障征兆集数据，归一化为网络输入模式。

② 建立神经网络系统，用已知故障征兆-故障模式的样本集训练网络，使其达到预设的诊断精度，得出标准故障模式。

③ 实时输入故障征兆向量进行测试，获得该状态下的网络输出模式，然后对网络输出进行后处理，再与标准故障模式进行对比，获得诊断结果，即故障类型。

16.4.2 问题实例

【例 16-3】 柴油机故障诊断实例。柴油机表面振动信号包含着丰富的工作状态信息和故障特征信息，从其表面振动信号中提取时域特征参数，可以有效地识别柴油机工作状态。通过实验采集振动信号作为识别故障的原始依据，建立基于振动信号的柴油机故障诊断神经网络，并对网络进行训练得出标准故障模式，运用该神经网络实现柴油机的故障诊断。

柴油机故障模式集包括 6 种状态，如表 16-10 所示。

表 16-10 柴油机故障模式集

编 号	故障状态
1	正常状态
2	第一缸喷油压力过大
3	第一缸喷油压力过小
4	第一缸喷油器针阀磨损
5	油路堵塞
6	供油提前角提前 5′ ~6′

在某类型柴油机上分别测定了 5 种柴油机燃油系统典型故障状态下和正常状态下的振动参数。选取 6 种时域特征参数构成神经网络的故障征兆输入样本集：能量指标（E）、峭度指标（Kv）、波形指标（S）、裕度指标（L）、脉冲指标（I）和峰值指标（C）。

在特定的工作条件下测量时域特征参数，测量得到的特征参数样本如表 16-11 所示。此处柴油机运行的工况条件为：900rpm，空载。其中每种状态下的样本 1、2 对应测点为缸盖的样本，而样本 3、4 则是测点为缸壁的样本。

表 16-11 输入样本

		能量指标 E	峭度指标 Kv	波形指标 S	裕度指标 L	脉冲指标 I	峰值指标 C
状态 1	样本 1	1.1803	10.4502	1.513	20.0887	15.465	10.219
	样本 2	1.2016	12.4476	1.555	20.6162	15.755	10.1285
	样本 3	3.0728	26.3167	1.816	32.0035	22.723	12.509
	样本 4	3.9688	33.6814	1.846	37.6196	26.61	14.4085
状态 2	样本 1	1.97	9.5332	1.534	16.7413	12.741	8.3052
	样本 2	1.234	9.8209	1.531	18.3907	13.988	9.1336

续表

		能量指标 E	峭度指标 Kv	波形指标 S	裕度指标 L	脉冲指标 I	峰值指标 C
	样本 3	1.9502	15.7767	1.597	24.5548	18.505	11.5849
	样本 4	1.3905	13.3733	1.589	20.1439	15.154	9.5324
状态 3	样本 1	0.7682	9.5489	1.497	14.7612	11,497	7.68
	样本 2	0.7053	9.5317	1.508	14.3161	11.094	7.3552
	样本 3	2.2764	20.6665	1.634	26.848	20.214	12.3662
	样本 4	1.5686	17.3158	1.627	22.1099	16.626	10.2175
状态 4	样本 1	0.8116	8.1302	1.482	14.3171	11.1105	7.4967
	样本 2	0.816	9.0388	1.497	15.0079	11.6242	7.7604
	样本 3	3.1503	30.077	1.719	36.8448	27.0329	15.719
	样本 4	4.3777	44.3974	1.733	42.8669	31.4872	18.1639
状态 5	样本 1	1.4311	8.9071	1.521	15.746	12.0088	7.8909
	样本 2	1.4136	8.6747	1.53	15.3114	11.6297	7.5984
	样本 3	2.9625	31.6339	1.727	31.4159	23.2644	13.4654
	样本 4	2.8378	30.5401	1.741	31.5176	23.2395	13.3435
状态 6	样本 1	1.167	8.3504	1.51	12.8119	9.8258	6.506
	样本 2	1.3392	9.0865	1.493	15.0798	11.6764	7.8209
	样本 3	3.3528	27.955	1.701	35.0426	25.7493	15.1333
	样本 4	3.7199	20.9654	1.726	33.4204	24.0614	13.9382

解： ① 对于缸盖和缸壁，需要建立不同的神经网络，首先建立缸盖部分网络，定义输入样本向量。输入命令：

```
P11=[1.1803 10.4502 1.5134 20.0887 15.4658 10.2193]';
P12=[1.2016 12.4476 1.5556 20.6162 15.7558 10.1285]';
P21=[1.0970 9.5332 1.5341 16.7413 12.7410 8.3052]';
P22=[1.2340 9.8209 1.5316 18.3907 13.9889 9.1336]';
P31=[0.7682 9.5489 1.4970 14.7612 11.4973 7.6800]';
P32=[0.7053 9.5317 1.5083 14.3161 11.0941 7.3552]';
P41=[0.8116 8.1302 1.4820 14.3171 11.1105 7.4967]';
P42=[0.8160 9.0388 1.4979 15.0079 11.6242 7.76041]';
P51=[1.4311 8.0971 1.5219 15.7460  12.0088  7.8909]';
P52=[1.4136 8.6747 1.5305 15.3114 11.6297 7.5984]';
P61=[1.0167 8.3504 1.5103 12.8119 9.8258 6.5060]';
P62=[1.3392 9.0865 1.4930 15.0798 11.6764 7.8209]';
P=[P11 P12 P21 P22 P31 P32 P41 P42 P51 P52 P61 P62];
```

以上则完成了输入样本向量 ***P*** 的定义。

② 对故障类型进行编码，如表 16-12 所示。

表 16-12 故障类型编码表

编 号	故障类型	输出向量
1	正常状态	(100000)
2	第一缸喷油压力过大	(010000)
3	第一缸喷油压力过小	(001000)
4	第一缸喷油器针阀磨损	(000100)
5	油路堵塞	(000010)
6	供油提前角提前 5'～6'	(000001)

期望输出为 6 维向量，其中故障类型位为 1 则代表发生了相应的故障。接下来定义期望输出向量。输入命令：

```
t11=[1 0 0 0 0 0]';
t12=[1 0 0 0 0 0]';
t21=[0 1 0 0 0 0]';
t22=[0 1 0 0 0 0]';
t31=[0 0 1 0 0 0]';
t32=[0 0 1 0 0 0]';
t41=[0 0 0 1 0 0]';
t42=[0 0 0 1 0 0]';
t51=[0 0 0 0 1 0]';
t52=[0 0 0 0 1 0]';
t61=[0 0 0 0 0 1]';
t62=[0 0 0 0 0 1]';
t=[t11 t12 t21 t22 t31 t32 t41 t42 t51 t52 t61 t62];
```

从而完成了期望输出向量的定义。

③ 仍旧采用 BP 神经网络进行故障诊断，此处定义 BP 网络参数如表 16-13 所示。

表 16-13 BP 神经网络参数设定

网络层数	各层神经元数目	传递函数	训练算法
3 层	[16, 6]	{'logsig','purelin'}	'trainlm'

定义 BP 神经网络，应用输入、输出样本向量，设定误差目标为 1e–4，对网络进行训练。输入命令：

```
net=newff(minmax(P),[16,6],{'logsig','purelin'},'trainlm')
net.trainParam.show = 100,
net.trainParam.epoch = 2000,
net.trainParam.goal= 1e-4,
[net,tr]=train(net,P,t),
```

执行上述命令，在弹出的 nntraintool 窗口中单击“Performance”按钮，绘出图形如图 16-6 所示。可以看到经过 17 次迭代网络就达到了期望误差目标。

④ 定义验证样本，对网络进行验证。验证样本如表 16-14 所示。

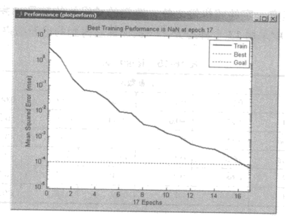


图 16-6 缸盖样本训练过程中的误差变化曲线

表 16-14 缸盖验证样本表

	E	K	S	L	I	C
样本 1	1.1833	11.8189	1.5481	20.2626	15.5814	10.0646
样本 2	1.2394	9.6018	1.5366	18.219	13.851	9.0142
样本 3	0.661	8.8735	1.508	13.598	10.5171	6.9744
样本 4	0.7854	8.7568	1.4915	14.4547	11.1971	7.5071
样本 5	1.2448	8.3654	1.5413	15.2558	11.5643	7.503
样本 6	1.3111	7.9501	1.4915	14.9174	10.7511	7.7127

输入命令：

```
ptest1 = [1.1833 11.8189   1.5481  20.2626 15.5814 10.0646]';
ptest2 = [1.2394 9.6018 1.5366  18.219  13.851  9.0142]';
ptest3 = [0.661 8.8735  1.508   13.598  10.5171 6.9744]';
ptest4 = [0.7854    8.7568  1.4915  14.4547 11.1971 7.5071]';
ptest5 = [1.2448    8.3654  1.5413  15.2558 11.5643 7.503]';
ptest6 = [1.3111    7.9501  1.4915  14.9174 10.7511 7.7127]';
ptest =[ptest1 ptest2 ptest3 ptest4 ptest5 ptest6 ];
result_test = sim(net, ptest)'
```

输出结果为：

```
result_test =
    0.8489    0.1729    0.0363   -0.0514   -0.0336    0.0233
   -0.0507    1.1214    0.0442   -0.0717    0.0443   -0.0963
   -0.0174    0.1247    1.1754   -0.2465   -0.0197   -0.0088
    0.0116   -0.0387    0.3135    0.7000   -0.0457    0.0627
    0.1108    0.3473   -0.3028    0.1217    1.2417   -0.3407
    0.1342   -0.3634   -0.3704    0.2865    0.3034    0.9778
```

将结果绘成表格，如表 16-15 所示。

可见，训练后的网络诊断结果是正确的。需要指出的是，运用 BP 网络进行故障诊断，

除了每次运行时都具有一定的随机性之外，选择的神经元的数目也会影响网络的诊断性能，如神经网络隐层中采用 13 个神经元，一次训练后得到的诊断结果如表 16-16 所示。

表 16-15 诊断结果 1

实际故障类型	诊断数据						诊断状态
类型 1	0.99	0.28	0.11	−0.06	−0.16	0.06	类型 1
类型 2	0.10	1.27	−0.01	−0.11	0.15	−0.25	类型 2
类型 3	−0.06	−0.51	0.74	0.62	0.36	−0.16	类型 3
类型 4	−0.08	−0.32	0.17	1.09	0.22	−0.12	类型 4
类型 5	0.47	0.68	0.01	−0.19	1.23	−0.70	类型 5
类型 6	0.23	−0.36	−0.26	0.49	0.18	1.02	类型 6

表 16-16 诊断结果 2

实际故障类型	诊断数据						诊断状态
类型 1	0.77	0.20	−0.02	0.05	−0.05	0.05	类型 1
类型 2	−0.01	0.98	−0.02	0.05	0.09	−0.16	类型 2
类型 3	−0.08	0.10	0.98	0.14	0.04	−0.11	类型 3
类型 4	−0.06	0.09	0.45	0.67	0.03	−0.11	类型 4
类型 5	0.07	−0.03	−0.30	0.60	1.06	−0.52	类型 5
类型 6	−0.07	−0.03	−0.55	0.91	0.09	0.78	类型 3

可以看到，隐层含 13 个神经元的 BP 神经网络将故障类型 6 的样本错误判决为故障类型 3，有兴趣的读者可以自行修改程序参数，并运行查看网络诊断性能。

⑤ 对于缸壁数据，采用同样的方法重新运行程序。训练过程中的误差变化曲线如图 16-7 所示。

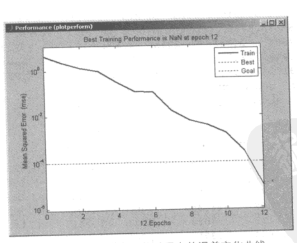


图 16-7 缸壁样本训练过程中的误差变化曲线

采用表 16-17 的验证样本集对网络进行验证。

表 16-17 缸壁验证样本集

	E	K	S	L	I	C
样本 1	2.6917	25.9008	1.8263	30.7798	21.721	11.893
样本 2	1.2558	12.2466	1.5869	19.3771	14.6	9.2006
样本 3	1.8559	18.3491	1.628	24.1876	18.164	11.157
样本 4	3.6823	37.713	1.7639	39.8645	31.393	16.493
样本 5	2.7157	29.5636	1.7245	30.7679	22.812	13.228
样本 6	3.2728	24.5571	1.7577	32.1747	23.226	14.214

最终得到的诊断结果如表 16-18 所示。

表 16-18 诊断结果

实际故障类型	诊断数据						诊断状态
类型 1	1.05	0.04	0.40	–0.10	0.33	–0.65	类型 1
类型 2	0.04	1.21	–0.20	0.00	–0.01	–0.02	类型 2
类型 3	0.04	0.10	0.97	0.04	–0.05	–0.08	类型 3
类型 4	–0.28	–0.12	–0.36	1.52	0.49	–0.33	类型 4
类型 5	–0.01	–0.04	0.00	0.02	1.01	0.01	类型 5
类型 6	0.10	0.07	0.20	0.13	–0.20	0.55	类型 6

可见，训练后的网络同样可以正确地进行缸壁的故障诊断。

16.5 基于神经网络的水循环系统故障诊断

16.5.1 问题背景

水循环系统是锅炉系统中的一个控制单元，用于对锅炉的用水供给和冷却，其组成结构如图 16-8 所示。

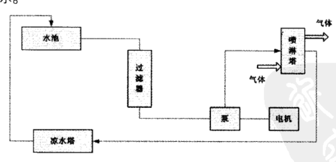


图 16-8 水循环冷却系统结构

如图 16-8 所示，贮存在沉淀水池中的水，经过过滤器将杂质过滤后送入离心泵的入口，流经泵体进行循环，离心泵则由电机进行驱动。

水循环冷却系统的工作流程是冷却水经离心泵加压后流至喷淋塔顶部的喷淋头，喷淋

头在此与自下而上的冷却气体进入热交换。冷却水经换热后温度升高，在塔内积存起来，并保持一定液位，防止气体逸出。塔底的排水管排出的冷却水进入凉水塔，在凉水塔中冷却水由风扇鼓出的风进行冷却后，回到沉淀池，整个水循环一周。

对于以上水循环系统构建一个故障诊断模块，自动地从该系统当前工艺参数中辨别和判断故障，既可避免控制人员对大量工艺参数进行检测，又可以使控制人员更全面地判断当前生产状态和预测将来的情况，从而及时采取有效应对措施。水循环系统的故障诊断对于现场控制人员来说是非常有用的。

BP 网络虽然可以以任意的精度逼近任何的非线性函数，但是 BP 神经网络存在一些缺陷：

（1）难以确定网络结构，需要通过多次试验进行优化和选取。

（2）存在局部极小点，无法保证网络最终收敛于全局最小点。

（3）收敛速度慢。对于非线性网络，采用基于梯度的学习算法时，网络可能出现麻痹现象，使网络的收敛速度变得非常缓慢。

（4）由于非线性网络存在局部极小点，权值的初始值可能会影响到网络最终的收敛性，实际应用中，通常采用给网络权值赋以小随机数的方法。

而 RBF 网络在以上这些方面相对于 BP 网络都有改善。此例中应用 RBF 径向基网络对此系统进行故障诊断。

16.5.2 问题实例

【例 16-4】 水循环系统故障诊断实例。通过传感器采集的实验数据，建立锅炉水循环系统故障诊断神经网络，运用该神经网络进行水循环系统的故障诊断。水循环系统故障类型集包含为 3 种状态，其编码如表 16-19 所示。

表 16-19 水循环系统故障类型集

编　号	故障类型	输出向量
1	泵体堵塞	(100)
2	机械故障	(010)
3	管道堵塞	(001)

采集参数包括检测点的进口压力（X1）、出口压力（X2）、出流量（X3）、轴承温度（X4），及电机电流（X5），以此实验数据加故障类型编码向量构成输入样本，如表 16-20 所示。此数据存于 Excel 文件 16_5_P.xls 中。

表 16-20 输入样本

序　号	故障类型	进口压力（X1）	出口压力（X2）	出流量（X3）	轴承温度（X4）	电机电流（X5）
1	泵体堵塞	0.83	0.66	0.83	0.78	0.76
2	泵体堵塞	0.83	0.66	0.85	0.8	0.76
3	泵体堵塞	0.83	0.55	0.8	0.8	0.76
4	泵体堵塞	1	0.4	0.66	0.82	0.78

续表

序　号	故障类型	进口压力（X1）	出口压力（X2）	出流量（X3）	轴承温度（X4）	电机电流（X5）
5	泵体堵塞	0.66	0.3	0.6	0.82	0.78
6	机械故障	0.83	0.84	1	0.95	0.88
7	机械故障	0.83	0.84	1	0.98	0.93
8	机械故障	1	0.78	0.92	1	0.9
9	机械故障	0.66	0.66	0.83	0.98	0.98
10	机械故障	0.83	0.62	0.85	0.98	1
11	管道堵塞	0.5	0.84	0.66	0.82	0.76
12	管道堵塞	0.2	0.66	0.6	0.84	0.78
13	管道堵塞	0.66	0.95	0.6	0.85	0.79
14	管道堵塞	0.66	1	0.5	0.9	0.78
15	管道堵塞	0.66	1	0.66	0.85	0.78

解： ① 定义输入样本向量。输入如下命令：

```
[filename, pathname] = uigetfile('16_5_P.xls');
file=[pathname filename];
x=xlsread(file);
p=x;
p=p';
```

② 定义期望输出向量，其中 3 种故障类型的编码格式如表 16-19 所示。在 MATLAB 命令空间中输入命令：

```
t1=[1 0 0]'
t2=[0 1 0]'
t3=[0 0 1]'
t = (t1 t1 t1 t1 t1 t2 t2 t2 t2 t2 t3 t3 t3 t3 t3);
```

③ 构建神经网络，此处采用径向基神经网络进行故障诊断。输入命令：

```
spread = 0.6;
net = newrbe(p,t,spread);
```

网络选用的散布常数值设定为 0.6。

④ 应用验证样本对此网络进行仿真测试，一共有 6 个测试样本，其定义如表 16-21 所示，这些数据存于 Excel 文件 16_5_Ptest 文件中。

表 16-21　测试样本

序　号	故障类型	进口压力（X1）	出口压力（X2）	出流量（X3）	轴承温度（X4）	电机电流（X5）
1	泵体堵塞	0.8	0.60	0.82	0.82	0.75
2	泵体堵塞	0.91	0.40	0.7	0.87	0.67
3	机械故障	0.64	0.30	0.6	0.8	0.73
4	机械故障	0.6	0.84	0.8	0.98	0.95
5	管道堵塞	0.8	0.8	0.92	1	0.98
6	管道堵塞	0.9	0.77	0.1	0.95	0.99

输入命令：

```
[filename, pathname] = uigetfile('16_5_Ptest.xls');
file=[pathname filename];
xtest=xlsread(file);
ptest=xtest;
ptest=ptest';
ans = sim(net, ptest),
```

输出结果为：

```
ans =
    1.1558    1.2558    0.1618    0.2195   -0.5328    0.1057
   -0.2332   -0.2197    0.4694    0.6702    0.7886    0.1362
    0.1773    0.0639    0.4688    0.2103    0.8443    0.8580
```

将预测结果列表，如表 16-22 所示。

表 16-22　测试样本

序　号	故障类型	输 出 1	输 出 2	输 出 3	诊断故障
1	泵体堵塞	1.16	–0.23	0.18	泵体堵塞
2	泵体堵塞	1.26	–0.22	0.06	泵体堵塞
3	机械故障	0.16	0.47	0.47	机械故障
4	机械故障	0.22	0.67	0.21	机械故障
5	管道堵塞	–0.53	0.79	0.84	管道堵塞
6	管道堵塞	0.11	0.14	0.86	管道堵塞

可以看到，上述构造的 RBF 神经网络能够用于此锅炉水循环系统的故障诊断。需要说明的是，散布常数是一个可调的值，对于不同的散布常数，生成的网络性能是不一样的。上面的散布常数 0.6 是经过尝试之后确定的值。

16.6 小结

本章首先概述了神经网络在故障诊断中的应用，然后结合 4 个大例子，综合讲述了 MATLAB 在神经网络故障诊断中的应用。

第 17 章　神经网络预测

预测是人们根据事物的发展规律、历史和现状，分析影响其变化的因素，对其发展前景和趋势预先进行的一种推测。

对于现实中大量存在的非线性、非平稳的复杂动力系统问题，非线性 ARMA 模型应用中需要确定合适的模型阶数，这是比较困难的，在传统的预测方法解决效果欠佳的领域，应用神经网络等智能预测方法往往能够更有效。

本章将对神经网络在不同领域内的预测应用通过实例进行说明和介绍，主要内容包括以下几部分。

（1）神经网络预测概述。

（2）基于神经网络的地震预测。

（3）基于神经网络的人口预测。

（4）基于神经网络的电信业务量预测。

（5）基于神经网络的股市预测。

（6）基于神经网络的公司财务风险预测。

17.1 神经网络预测概述

神经网络中单个神经元具有简单的能够反映非线性本质特征的能力，通过这些基本的单元自组织复合，使神经网络能够重建任意的非线性连续函数。通过学习这一归纳过程，可以使网络获得序列的内在规律，从而可以对序列的变化进行预测。使用神经网络方法避免了烦琐的常规建模过程，而且神经网络模型有良好的自适应和自学习能力、较强的抗干扰能力，易于给出工程上容易实现的算法。用于预测领域的神经网络通常有：BP 神经网络、径向基函数神经网络、Elman 神经网络等。

神经网络应用于预测，大体可以分为两种方式。

1. 基于回归分析的神经网络预测

回归分析预测法是利用因素之间的因果关系，通过建立回归网络进行预测。该方法具有使用方便、能进行长期预测，以及预测精度较高的特点。

基于回归分析的神经网络预测方法的步骤如下。

① 分析变量的影响因素，并确定出主要因素及其影响程度。

② 利用历史数据建立预测变量与主要影响因素之间的回归预测网络。

③ 把预测期各主要影响因素的指标值，代入网络进行预测。

2. 基于时间序列的神经网络预测

基于时间序列的神经网络预测模型的建模过程与基于回归分析的建模过程基本相同，只是不需要确定影响因素，因此在对输入神经元的确定上也有所不同。

在样本数据确定的情况下，输入神经元的数目过多，则供网络训练的学习样本太少；反之，输入神经元的数目过少，则学习样本太多。该方法的优点是没有复杂的理论问题，简便易行，但是缺少理论依据。

自从 1987 年 Fabrer 首先应用神经网络进行预测以来，基于神经网络的时间序列预测方法受到重视。目前，已有多种不同形式的神经网络被用于工业、经济等的预测中。研究结果表明，神经网络用于预测的效果较好，为一类高度非线性动态关系的时间序列预测提供了一条有效途径。

可以利用已知数据建立一系列准则用于预测。我们假设一次观测中过去值与未来值之间存在联系，找到一个函数，当过去观测值作为输入时，给出未来值作为输出，这个模型是由神经网络来实现的。

相对传统的统计预测方法而言，神经网络模型有良好的非线性品质、灵活而有效的学习方式，以及完全分布式的存储结构。

在时间序列预测中，前馈网络是最常使用的网络。在这种情形下，从数学角度看，网络成为非线性函数。记一个时间序列为$\{X_n\}$，对其进行预测可用下式描述：

$$X_{n+k}=f(X_n,X_{n-1},X_{n-2},\dots,X_1)$$

时间序列预测方法即用神经网络来拟合函数 f，然后预测未来值。这就是利用神经网络进行时间序列预测的基本思路。

一般将外生变量样本观察值或时间序列数据作为网络的输入向量，将内生变量样本值作为输出，通过训练网络，可以预测内生变量的下一步走向。无论系统模型是何种类型，用于表述这些神经网络的框架结构是不变的。

尽管神经网络也存在一些不足，比如缺乏统计合理性，不易解释，缺乏稳健性，运算速度慢，隐层节点数依赖经验等，但它突破了传统预测方法的许多局限性，在实践中有着较为广泛的应用，是一种很有前景的预测方法。

利用神经网络进行预测，首先要对系统建模。系统建模通常有两种方式：正向建模和逆向建模。

1. 正向建模

正向建模是指训练一个神经网络表达系统正向动态的过程，这一过程建立的神经网络模型称为正向模型，其结构如图 17-1 所示。其中神经网络与待辨识系统并联，两者输出的误差作为网络的训练信号。显然，这是一个典型的有监督学习过程，实际系统作为导师，向神经网络提供训练算法所需的期望输出。

比如建模系统为被控对象时，神经网络多采用多层前向网络，可直接选择误差反向传播算法或其变形，这时神经网络代替被控对象，用来提供控制误差的反向传播通道，或直

接替代传统控制器，如 PID 控制器等。而当系统为性能评价器时，可选择再励学习算法。这里的网络既可以采用有全局逼近能力的神经网络，如多层感知器，也可以选用有局部逼近能力的神经网络，如小脑模型关节控制器（CMAC）等。

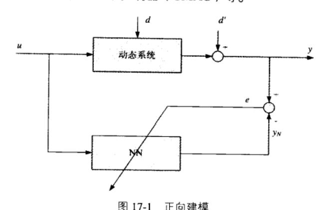


图 17-1　正向建模

2. 逆向建模

逆向建模又分为直接逆建模和正-逆建模，其中，直接逆建模的结构如图 17-2 所示。

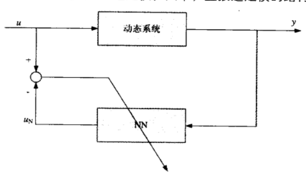


图 17-2　直接逆建模

直接逆建模也称为广义逆学习（Generalized Inverse Learning），从原理上来说是一种简单的方法。从图中可以看出，待辨识系统的输出作为网络的输入，网络输出与系统输入比较，相应的输入误差用来进行训练，因而网络将通过学习建立系统的逆模型。但是如果所辨识的非线性系统是不可逆的，利用上述方法，可能得不到一个正确的逆模型。因此，在建立系统的逆模型时，必须事先确定系统是可逆的。

正-逆建模也称为狭义逆学习（Specialized Inverse Learning），其结构如图 17-3 所示。这时待辨识的网络位于系统之前，并与之串联。网络的输入为系统的期望输出 $y_d(t)$，训练误差为期望输出与实际输出之差，或为期望输出与已建立的神经网络正向模型输出之差。

此方法的特点是，通过使用系统已知的正向动力学模型，或增加使用已建模的神经网络正向模型，以避免再次采用系统输入作为训练误差，使待辨识的网络仍然沿期望输出进

行学习，从根本上克服了使用系统输入作为训练误差所带来的问题。此外，对于不可逆的系统，利用此方法也可以得到一个具有期望特性的特殊逆模型。

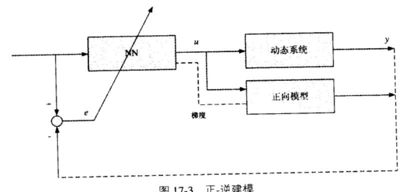


图 17-3　正-逆建模

17.2 基于神经网络的地震预测

17.2.1 问题背景

地震的发生会给社会带来不同程度的破坏，造成生命、财产的损失。目前地震学只是一门观测科学，人们只能加强观测，通过各种经验方法来进行预测，确定重点监测区。

地震预测是对未来破坏性地震发生的时间、地点和震级及地震影响的预测，是根据地区地质、地震活动性、地震前兆异常和环境因素等多种手段的研究与前兆信息监测所进行的现代减灾科学。地震预测技术是从地震监测、大震考察、野外地质调查、地球物理勘探、室内实验等多方面对地震发生的条件、规律、前兆、机理进行预报，并研究其方法及对策的综合技术。

由于地震学指标与地震之间存在较强的不确定性，通常一种地震学指标在一次地震中出现，而在其他的地震中不出现或不明显，因此，单一使用某一个地震学指标的预测方法就存在局限性。而如何解决预测结论的权值以及各指标组合时相互间的非线性联系对最终预测结论产生影响，就成为难题，神经网络技术的发展给我们提供了一个解决这个问题较好的方法。

神经网络相对于其他预测手段，具有并行分布处理与存储、高度容错、自组织、自适应和自学习功能，能分析较为复杂的非线性系统。它通过分析各个指标，建立其内在联系，并根据这种联系对目标进行预测，能有效地避免了采用单一指标进行预测的片面性，大大提高了预测的精度。

下面就 Elman 神经网络算法及其在地震预测中的具体运用实例进行介绍。

17.2.2 问题实例

【例 17-1】 地震预测实例。选取某地沿海东经 117° ~120° ，北纬 22° ~26° 范围内的地震在时间、空间、强度三方面具有代表性的 6 个指标：频度、蠕变、能量、B 值、缺

震、η 值指标的 29 组数据（如表 17-1 所示）作为网络的输入，利用神经网络对其第二年可能发生的地震最大震级进行预测。

表 17-1 地震样本数据

频 度	蠕 变	能 量	B 值	缺 震	η 值	期望输出
0.144	0.5879	0.2937	0.71	0.7778	0.598	4
0.15	0.399	0.2314	0.44	1	0.809	4.1
0.19	0.7596	0.368	0.62	0.8283	0.8542	4.1
0.271	0.103	0.1452	0.59	0.8754	0.995	3.9
0.23	0.5343	0.2183	0.7	0.7946	1	4
0.188	0.6384	0.8646	0.66	0.8114	0.9648	4.8
0.204	0.6697	0.9105	0.62	0.8283	0.9196	4.3
0.157	0.8152	0.5797	0.47	0.936	0.8693	3.8
0.211	0.7364	0.3865	0.69	0.8182	0.8291	4.4
0.213	0.7808	0.1648	0.63	0.8081	0.8342	4
0.126	0.3737	0.1823	0.75	0.7576	0.6935	5
0.125	0.199	0.132	0.69	0.6936	0.7437	3.9
0.126	0.2939	0.5895	0.75	0.7374	0.7136	3.7
0.073	0.9414	0.4258	0.84	0.633	0.794	4
0.103	0.1778	0.2074	0.75	0.7104	0.8291	3.8
0.156	0.5768	0.56	0.7	0.7879	0.7839	5.5
0.159	0.8515	0.13	0.54	0.8586	0.8995	3.9
0.177	0.2798	0.3111	0.76	0.724	0.8995	3.9
0.135	0.2859	0.5087	0.71	0.7441	0.8744	4.1
0.094	0.5687	0.1714	0.55	0.8165	0.9096	3.9
0.211	0.6899	0.4279	0.74	0.8081	0.7337	4.9
0.12	0.4899	0.3504	0.56	0.8215	0.9899	3.9
0.154	0.6828	0.0156	0.67	0.7946	0.7889	4.4
1	1	0.394	0.7	0.983	0.6784	5.5
0.33	0.1222	0.8603	0.68	0.8788	0.7186	4.7
0.163	0.3162	0.2402	0.87	0.7003	0.6834	4.1
0.771	0.9121	1	1.03	0.7576	0.7437	5.1
0.238	0.1091	0.3734	0.69	0.8384	0.7136	4.3
0.785	0.3512	0.0292	0.85	0.9339	0.7867	4.6

表中，前 6 列为学习样本中的输入样本，而第 7 列则为期望响应，是这一地区第二年可能发生的地震的最大震级。

解： ① 定义期望样本向量。在 MATLAB 命令空间中输入命令：

```
[filename, pathname] = uigetfile('12_1_P.xls');
file=[pathname filename];
x=xlsread(file);
p=[x(:,1),x(:,2),x(:,3),x(:,4),x(:,5),x(:,6);]'
t= x(:,7)',
```

接着定义测试样本向量。输入命令：

```
[filename, pathname] = uigetfile('12_1_Ptest.xls');
file=[pathname filename];
x=xlsread(file);
ptest=[x(:,1),x(:,2),x(:,3),x(:,4),x(:,5),x(:,6);]'
ttest=x(:,7)'
```

② 创建 Elman 神经网络，并对其进行训练。Elman 神经网络是一种带反馈的两层神经网络，反馈连接从第一层输出连接到输入端，此反馈连接使得 Elman 网络能够检测和生成时变模式。输入命令：

```
net=newelm(minmax(p),[25,1],{'logsig','purelin'}),
net.trainParam.show = 200,
net.trainParam.epoch = 3000,
net.trainParam.goal= 1e-2,
[net,tr]=train(net,p,t);
```

此 Elman 网络结构可以由弹出的 nntraintool 窗口上的结构图看出，如图 17-4 所示。

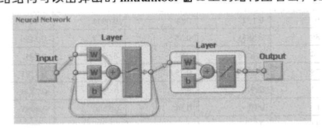


图 17-4 Elman 网络结构

从图中可以看到，Elman 网络的隐层也即反馈层的神经元采用 logsig 传递函数，而输出层采用线性传递函数。网络误差变化如图 17-5 所示，就训练情况来说，网络实际上并没有达到 1e–2 的误差目标，但仍然可以尝试将训练后的网络应用于预测。

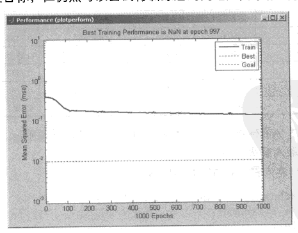


图 17-5 训练过程中网络误差的变化

③ 训练完毕后，首先将所有样本输入网络，然后定义检验向量，并将检验向量输入网络，查看输出值和输出误差。输入命令：

```
rtest = sim(net, p);
rdelta = rtest-t;
result_test = sim(net, ptest);
delta = result_test-ttest;
result = [ttest' result_test' delta']
```

输出结果为：

```
result =
    4.9000    4.6375   -0.2625
    3.9000    3.9048    0.0048
    4.4000    4.0387   -0.3613
    5.5000    4.0645   -1.4355
    4.7000    4.5633   -0.1367
    4.1000    4.4539    0.3539
    5.1000    4.4907   -0.6093
    4.3000    4.3753    0.0753
    4.6000    4.1862   -0.4138
```

将结果显示为表格，如表 17-2 所示。

表 17-2 期望输出与实际输出比较

序 号	期望输出	实际输出	预测误差
1	4.9	4.6375	-0.2625
2	3.9	3.9048	0.0048
3	4.4	4.0387	-0.3613
4	5.5	4.0645	-1.4355
5	4.7	4.5633	-0.1367
6	4.1	4.4539	0.3539
7	5.1	4.4907	-0.6093
8	4.3	4.3753	0.0753
9	4.6	4.1862	-0.4138

可以看到，对第 4 个和第 7 个样本，预测的偏差较大。以预测震级偏离期望值 0.5 为标准，则此 Elman 网络的预测正确率为 77%。

在用神经网络进行预测时，得出的结果体现了选定的各个指标的“集体”智慧，大大提高了预测的准确度。但从神经元的选取、层面的确定和迭代次数的变化等因素来看，神经网络的预测效果也是有很大的主观随意性的，神经网络模型会因选取的输入、训练次数的不同而得出不同的结果，这也反映出现阶段地震预测水平的一个共性，即任何预测方法都带有局限性。

尤其是在神经网络方法中，一方面我们选用这些地震学指标仅是地震活动的表象的指标，而不是目前也还不可能是直接反映发震物理实质的指标（例如震源处的应力大小、破裂类型等指标）；另一方面，选取的指标之间还很难满足输入元相互完全的“独立性”要求。但是，由于神经网络的“自学习”过程能较好地调整各指标对地震预测结果的权值，

使得这一方法能够得到较理想的预测效果，不失为一种可选择的数学模型。

17.3 基于神经网络的人口预测

17.3.1 问题背景

世界人口的迅猛增长引发了许多问题。特别是一些经济不发达国家的人口过度增长，影响了整个国家的经济发展、社会安定和人民生活水平的提高，给世界发展带来许多问题。

为了解决人口增长过快的问题，人类必须控制人口的增长，使之与社会、经济的发展相适应，与环境、资源相协调。

人口预测对于提供准确的人口信息，对于国家制定生育政策和社会发展计划有重要的意义。目前国内外提出了很多种预测方法和预测模型，如常微分方程法，动态预测法，AR 方法等。在这些方法中，由于神经网络方法具有识别复杂非线性系统的特性，具有高速的自学习、自适应能力，因而获得了较为广泛的应用。

17.3.2 问题实例

【例 17-2】 人口预测实例。应用 BP 神经网络建立时间序列人口预测模型，检验预测效果，我国 1994-2005 年人口数据如表 17-3 所示。

表 17-3 1994-2005 年人口数据

年份	人数（亿）	年份	人数（亿）
1994	11.9850	2000	12.6743
1995	12.1121	2001	12.7627
1996	12.2389	2002	12.8453
1997	12.3626	2003	12.9227
1998	12.4810	2004	12.9905
1999	12.5768	2005	13.0756

BP 神经网络的输入层、隐含层和输出层节点数取决于对象的复杂性。本例中对于 1994-2002 年的数据，选择前 3 年的数据作为序列的输入样本，下一年数据作为网络的输出；选用 2004、2005 年的数据为检验样本，如表 17-4 所示。

表 17-4 样本选择

训练样本序号	输 入	输 出
1	1994-1996 年的人口数	1997 年的人口数
2	1995-1997 年的人口数	1998 年的人口数
3	1996-1998 年的人口数	1999 年的人口数
4	1997-1999 年的人口数	2000 年的人口数
5	1998-2000 年的人口数	2001 年的人口数

续表

训练样本序号	输　入	输　出
6	1999-2001 年的人口数	2002 年的人口数
7	2000-2002 年的人口数	2003 年的人口数
验证样本	输　入	输　出
1	2001-2003 年的人口数	2004 年的人口数
2	2002-2004 年的人口数	2005 年的人口数

解： ① 定义期望样本向量。在 MATLAB 命令空间中输入命令：

```
p0=[11.9850 12.1121 12.2389 12.3626 12.4810 12.5768...
 12.6743 12.7627 12.8453 12.9227 12.9905 13.0756];
for i =1:7
   p(:,i)=[p0(i) p0(i+1) p0(i+2)]';
   t(i)=p0(i+3);
end
p
t
```

输出为：

```
p =11.9850   12.1121   12.2389   12.3626   12.4810   12.5768   12.6743
   12.1121   12.2389   12.3626   12.4810   12.5768   12.6743   12.7627
   12.2389   12.3626   12.4810   12.5768   12.6743   12.7627   12.8453
t =12.3626   12.4810   12.5768   12.6743   12.7627   12.8453   12.9227
```

② 构建 BP 网络，设置为隐层神经元为 5 个，输出层神经元为 1 个，并利用输入样本向量进行训练。输入命令：

```
net=newff(minmax(p),[5,1],{'logsig','purelin'},'trainlm'),
net.trainParam.show = 100,
net.trainParam.epoch = 2000,
net.trainParam.goal= 1e-4,
[net,tr]=train(net,p,t);
```

训练过程中，网络误差的变化情形如图 17-6 所示。

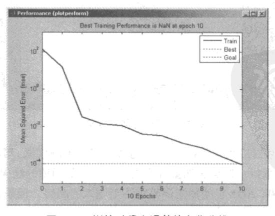


图 17-6　训练过程中误差的变化曲线

经过 10 次迭代，网络达到了期望的误差目标。

③ 训练完毕后，首先将所有样本输入网络，然后定义检验向量，并将检验向量输入网络，查看输出值和输出误差。输入命令：

```
ptest(:,1)=[p0(8) p0(9) p0(10)]';
ptest(:,2)=[p0(9) p0(10) p0(11)]';
ttest(1)=p0(11);
ttest(2)=p0(12);
result_test = sim(net,p)
result_test1 = sim(net,ptest)
delta = result_test1-ttest
```

输出结果为：

```
    result_test =12.3738   12.4670   12.5891   12.6710   12.7588   12.8526
12.9130
    result_test1 =    12.9604   12.9891
    delta =   -0.0301   -0.0865
```

可以看到，对于训练样本，网络输出与期望响应符合得很好，检验样本预测值误差分别为–0.0301 和–0.0865。绘图显示预测曲线与真实曲线。输入命令：

```
result = [result_test result_test1];
plot([1997:2005],p0(4:12),'-+',[1997:2005],result,'-go')
```

绘出曲线如图 17-7 所示。

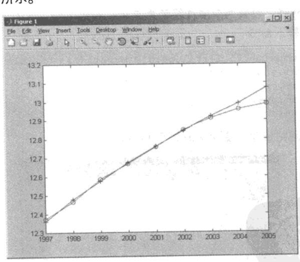


图 17-7　预测曲线与真实数据

其中加号“+”曲线对应为实际数据，“o”曲线对应为预测数据。可以看到，在 1997-2003 年之间两条曲线符合很好，这是在期望目标响应之下监督训练的结果，而这之后预测值则逐渐偏离真实值，2004-2005 年的预测误差有逐渐增大的趋势，但是误差大小在 0.1 以内，说明此网络的预测精度仍然是较高的。

17.4 基于神经网络的电信业务量预测

17.4.1 问题背景

电信业是国民经济的一个重要组成部分。电信业务量是表示需要传递的电信信息数量，它是核算设备、生产人员和组织生产的主要依据。

电信业务量与电信业务收入在以货币计量的数量上是不相等的，电信业务量仅要计算计费业务部分，而且还要计算免费业务部分。因为免费业务只是没有取收入，但是它和计费业务一样具有使用价值，电信企业需要为免费业务提供和计费业务相同的服务。因此，电信业务量更能完整地反映电信企业的业务种类和业务量。

电信业务量的预测，是编制电信专业网络规划工程可行性研究，乃至初步设计的要内容，预测结果是否符合客观实际，能否正确反映未来的发展趋势，直接关系拟建设的电信网络结构、工程规模、投资的大小以及经济效益的好坏，因此是一项重要的基建工作。

17.4.2 问题实例

【例 17-3】 电信业务量预测实例。应用 BP 神经网络进行电信业务量预测，并检验预测效果，我国 1989-2005 年电信业务量如表 17-5 所示。

表 17-5 1989-2005 年电信业务量数据

年　份	业务量（万户）	年　份	业务量（万户）
1989	123.46	1998	2431.21
1990	155.54	1999	3330.82
1991	204.38	2000	4792.70
1992	290.94	2001	4556.26
1993	462.71	2002	5695.80
1994	688.19	2003	7019.79
1995	988.85	2004	9712.29
1996	1342.04	2005	12028.54
1997	1773.29		

BP 神经网络的输入层、隐含层和输出层节点数取决于对象的复杂性。这里对于 1994-2002 年的数据，选择前 4 年的数据作为序列的输入样本，下一年数据作为网络的输出；选用 2004、2005 年的数据为检验样本，如表 17-6 所示。

表 17-6 样本数据

训练样本序号	输　入	输　出
1	1989-1992 年的业务量	1993 年的业务量
2	1990-1993 年的业务量	1994 年的业务量
3	1991-1994 年的业务量	1995 年的业务量

续表

训练样本序号	输　入	输　出
4	1992-1995 年的业务量	1996 年的业务量
5	1993-1996 年的业务量	1997 年的业务量
6	1994-1997 年的业务量	1998 年的业务量
7	1995-1998 年的业务量	1999 年的业务量
8	1996-1999 年的业务量	2000 年的业务量
9	1997-2000 年的业务量	2001 年的业务量
10	1998-2001 年的业务量	2002 年的业务量
11	1999-2002 年的业务量	2003 年的业务量
验证样本	输　入	输　出
1	2000-2003 年的业务量	2004 年的业务量
2	2001-2004 年的业务量	2005 年的业务量

解： ① 定义训练样本向量。在 MATLAB 命令空间中输入命令：

```
p0 =[123.46 155.54 204.38 290.94 462.71 688.19 988.85 1342.04...
1773.29 2431.21 3330.82 4792.7 4556.26 5695.8 7019.79 9712.29 12028.54];
```

由于输入向量中的数据元素变动范围很大，所以这里首先将其进行归一化处理，将其转化为[0,1]区间的实数。输入命令：

```
a=max(p0);
b=min(p0);
for i=1:17
p0(i) = (p0(i)-b)/(a-b);
end
```

以上完成了输入向量的归一化处理，下面对输入向量进行定义。输入命令：

```
for i = 1:11
   p(:,i)=[p0(i) p0(i+1) p0(i+2) p0(i+3)]';
   t(i) = p0(i+4);
end
```

完成训练样本向量定义。

② 构建 BP 神经网络，设置网络隐层神经元数目为 11 个，并应用样本向量进行训练。输入命令：

```
net=newff(minmax(p),[11,1],{'logsig','purelin'},'trainlm'),
net.trainParam.show = 100,
net.trainParam.epoch = 2000,
net.trainParam.goal= 1e-4,
[net,tr]=train(net,p,t);
```

网络经过 3 次迭代就完成训练，其误差达到误差目标以下。训练过程中网络误差如图 17-8 所示。

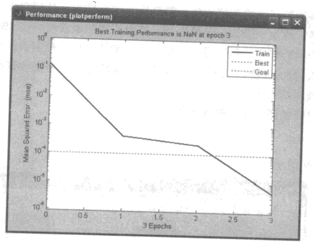


图 17-8 预测曲线与真实数据

③ 定义测试样本向量。输入命令：

```
ptest(:,1)=[p0(12) p0(13) p0(14) p0(15)]';
ptest(:,2)=[p0(13) p0(14) p0(15) p0(16)]';
ttest(1)=p0(16);
ttest(2)=p0(17);
```

④ 对网络进行仿真，并将期望输出、预测结果以及预测误差组合输出。输入下列命令：

```
result_test = sim(net,p)
result_test1 = sim(net,ptest)
result = [result_test result_test1];
ttest2 = [t ttest];
result2 = [ttest2' result' (result-ttest2)']
```

输出结果为：

```
result2 =
   0.0285    0.0334    0.0049    0.1709
   0.0474    0.0509    0.0034    0.0723
   0.0727    0.0721   -0.0006   -0.0085
   0.1024    0.1039    0.0016    0.0154
   0.1386    0.1425    0.0040    0.0286
   0.1938    0.1943    0.0005    0.0025
   0.2694    0.2667   -0.0028   -0.0102
   0.3922    0.3797   -0.0125   -0.0319
   0.3723    0.3662   -0.0062   -0.0165
   0.4681    0.4693    0.0013    0.0027
   0.5793    0.5797    0.0004    0.0007
   0.8054    0.8430    0.0376    0.0467
   1.0000    1.2498    0.2498    0.2498
```



其中，第一列为期望输出，第二列为实际输出，第三列为绝对预测误差，第四列为相

对预测误差。除最后两行为检测样本外，其余均为训练样本及其预测输出值。可以看到，对于训练样本向量，除第一个样本向量之外，其余相对预测误差均在 8%以下。

对于最后两个检验样本，可以看到，第一个检测样本相对预测误差为 4.67%，而第二个样本误差则大大增加，达到了 24.98%；这说明，该网络进行外推计算第一步是比较准确的，但是第二步预测就不够准确了。

下面对结果进行绘图显示，输入下列命令：

```
plot([1993:2005],p0(5:17),'-r*',[1993:2005],result,'-o')
```

绘出曲线如图 17-9 所示。

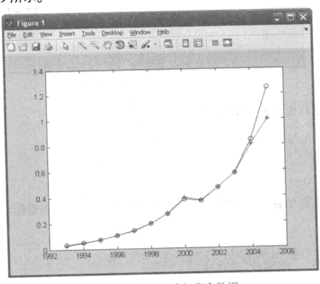


图 17-9 预测曲线与真实数据

17.5 基于神经网络的股市预测

17.5.1 问题背景

股票市场是国民经济的“晴雨表”和“报警器”，它不仅被政府所重视，更受到广大投资者的关注。

神经网络可以通过学习掌握数据之间的依存关系，进行股市预测。与以往的传统方法相比，这种方法具有很大的优越性。股价走势呈高度非线性，并且成交价、成交量中包含有大量决定股价变动的内在规律和特点，通过对历史交易数据的学习，神经网络能够从复杂的数据中找出参数之间的规律。神经网络是通过反复对照观测样本，逐步逼近理想的结果的。一个以线性函数作为转换函数的前向二层神经网络相当于一个线性回归模型；当转换函数为非线性时，一个前向多层网络相当于一个复杂的非线性回归模型。

对于非线性回归问题，人们通常用变量的变换转化为线性回归问题。这种变换依赖于

建模者的观察和经验，在模型涉及多个自变量时，要找到恰当的变换并不容易，而神经网络通过自我学习过程寻找变量之间的规律，具有很强的适应能力。同时它也不像线性回归那样对误差项的统计分布有严格的要求，还可以处理不完全的数据。

17.5.2 问题实例

【例 17-4】 股市预测实例。选择 2000 年某一时段每日股市收盘价作为样本，应用 BP 神经网络对其进行时间序列预测。基于大多数的股票交易者都是中线或者长线持有股票以获取最大的利润这一事实，此处主要进行中期预测的研究。将股票收盘价数据按周取平均值，得到该周的平均收盘价，如表 17-7 所示。

表 17-7 样本数据

序 号	样 本 值	序 号	样 本 值
1	1258.02	16	1651.22
2	1267.43	17	1713.81
3	1294.54	18	1708.22
4	1298.66	19	1666.55
5	1334.43	20	1675.05
6	1354.66	21	1597.08
7	1392.62	22	1583.72
8	1416.8	23	1597.37
9	1542.83	24	1614.59
10	1635.86	25	1650.4
11	1611.7	26	1664.58
12	1660.15	27	1694.19
13	1619.23	28	1733.39
14	1548.22	29	1731.1
15	1595.7	30	

解： ① 定义训练样本向量。在 MATLAB 命令空间中输入命令：

```
p0=[1258.02 1267.43 1294.54 1298.66 1334.43 1354.66 1392.62 1416.8 1542.83 ...
    1635.86 1611.7 1660.15 1619.23 1548.22 1595.7 1651.22 1713.81 1708.22...
    1666.55 1675.05 1597.08 1583.72 1597.37 1614.59 1650.4 1664.58 1694.19...
    1733.39 1731.1];
a=max(p0);
b=min(p0);
for i=1:29
    p0(i)=(p0(i)-b)/(a-b);
end
for i = 1:22
    p(:,i)=[p0(i); p0(i+1); p0(i+2); p0(i+3)];
    t(i) = p0(i+4);
end
```

于是训练样本向量定义完毕。

② 构建 BP 神经网络，设置网络隐层神经元数目为 12 个，接下来应用训练样本对网络进行训练。输入命令：

```
net=newff(minmax(p),[12,1],{'logsig','purelin'},'trainlm'),
net.trainParam.show = 100,
net.trainParam.epoch = 2000,
net.trainParam.goal= 1e-4,
[net,tr]=train(net,p,t);
```

训练迭代 5 次之后即达到误差性能目标。单击弹出的 nntraintool 窗口中的“Performance”按钮，绘出误差性能变化曲线，如图 17-10 所示。

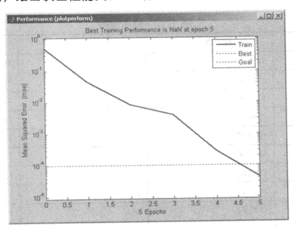


图 17-10 训练过程中的误差变化曲线

③ 定义测试样本向量，对网络进行仿真，并将期望输出、预测结果以及预测误差组合输出。输入命令：

```
ptest(:,1)=[p0(23) p0(24) p0(25) p0(26)]';
ptest(:,2)=[p0(24) p0(25) p0(26) p0(27)]';
ptest(:,3)=[p0(25) p0(26) p0(27) p0(28)]';
ttest(1)=p0(27);
ttest(2)=p0(28);
ttest(3)=p0(29);
result_test = sim(net,p);
result_test1 = sim(net,ptest)
delta = result_test1-ttest
result = [result_test result_test1];
plot([1:25],p0(5:29),'-r*',[1:25],result,'-o')
```

输出结果为：

```
result_test1 =    0.7813    0.9211    1.0562
delta =   -0.1363   -0.0789    0.0611
```

三点的相对预测误差为[−14.85%　7.89%　6.14%]。绘出的预测曲线如图 17-11 所示，

其中用“*”画出的曲线代表实际值，用“o”画出的曲线代表预测值，最后的三点为预测检验样本点。

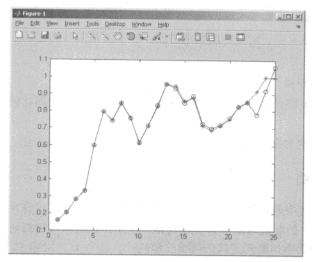


图 17-11 预测曲线与真实数据

由图可见，网络预测具有一定的误差，并不能够保证每一个点的预测值和股价变动趋势都与实际值相符。若是采用更大容量的样本，可能会得到更好的预测精度，这是需要进一步研究的问题。

17.6 基于神经网络的信用风险预测

17.6.1 问题背景

信用风险又称违约风险，是指借款人、证券发行人或交易对方因种种原因不愿或无力履行合同条件而构成违约，致使银行、投资者或交易对方遭受损失的可能性。对于上市公司而言，这种违约行为经常表现为拖欠账款、资不抵债，以及以发行证券或债券进行圈钱的失信行为。对这种违约失信的可能性的度量显得十分重要，建立上市公司信用风险评价模型是防范信用风险的重要手段。

国外对于信用风险预测模型的研究相对比较成熟，在统计方法运用方面主要有以下几个模型。

（1）单变量分析法

单变量分析是最早应用于财务困境预测的模型，其主要思想是通过比较财务困境企业和非财务困境企业之间某个财务指标的显著差异，从而对财务困境企业提出预警。

（2）多元判别分析法

为了克服单变量分析的局限性，通过多元判别模型产生了一个总的判别分，称为 Z 值，

并依据 Z 值进行判断。该模型也成为 Z 模型。Z 模型在破产前 1 年预测精确度达到 95%，前 2 年精确度达到 72%，但是从前 3 年起精确度就小于 50%了。

（3）Logistic 回归模型

Logit 分析广泛应用于财务困境预测方面。许多国家的研究者都分别建立了适合本国情况的 Logit 模型。

（4）神经网络模型

运用 BP 神经网络的对财务危机进行预测。结果发现，神经网络模型的效果要明显好于判别分析模型。

运用神经网络的对财务危机进行预测，是一种可行、有效的方法。

17.6.2 问题实例

【例 17-5】 信用风险预测实例。以 2007 年和 2008 年我国沪深两市上市公司中首次被 ST（特别处理）的制造业上市公司为对象，应用径向基概率神经网络进行信用风险预测研究，将其划分为有风险和无风险两类。经统计，2007 年首次被 ST 的制造业上市公司有 18 家，2008 年首次被 ST 的制造业上市公司有 20 家。现在根据第（t–3）年上市公司的年度财务数据来预测在第 t 年是否成为 ST 公司。

在 18 家 2007 年首次被 ST 的制造业上市公司选取 15 家、在 20 家 2008 年首次被 ST 的制造业上市公司中选取 15 家，即共 30 家作为研究样本，将这 30 家配对的正常公司作为配对样本。剩余的 8 家 ST 公司与 8 家正常公司作为验证样本，如表 17-8 所示。由于数据量较多，故此处数据没有完全列出，详细数据见本章光盘附带的 ST30、NST30、ST08、NST08 四个 Excel 文件。

表 17-8 部分样本数据

证券代码	资产报酬率	总资产净利润率	……	每股收益	每股营业收入	每股净资产
000403	-0.08143	-0.10428		-1.07308	3.164212	2.551181
000408	-0.007087	-0.03303		-0.32309	1.866199	3.666225
000631	-0.155459	-0.18834		-2.22178	3.367119	0.333422
000719	-0.038932	0.001122		0.006646	1.663097	2.187275
000732	-0.050868	-0.08843	……	-0.53519	4.067818	2.011746
000757	-0.002219	-0.03411		-0.22662	4.723867	1.866474
000925	-0.000099	-0.03431		-0.22279	1.929084	4.031781
000979	-0.277833	-0.30516		-3.09592	3.736802	1.649247
600065	-0.153102	-0.22282		-0.99745	2.064274	1.579475

解： ① 定义训练样本向量 $\boldsymbol{p}$，$\boldsymbol{T}$ 以及检验样本向量 ptest。其中 $\boldsymbol{p}$ 与 $\boldsymbol{T}$ 前 30 列为 ST 公司各项财务指标，后 30 列对应非 ST 公司各项财务指标；ptest 前 8 列对应 ST 公司各项财务指标，后 8 列对应非 ST 公司各项财务指标。

在 MATLAB 命令空间中输入命令：

```
[filename, pathname] = uigetfile('ST30.xls');        %读入 30 家 ST 公司数据
file=[pathname filename];
x=xlsread(file);
[filename, pathname] = uigetfile('NST30.xls');       %读入 30 家非 ST 公司数据
file=[pathname filename];
y=xlsread(file);
p=[x;y];                                             %定义训练输入向量
p=p';
T=[ones(30,1);zeros(30,1)];
T=T';
[filename, pathname] = uigetfile('ST08.xls');        %读入 8 家 ST 公司数据
file=[pathname filename];
x1=xlsread(file);
[filename, pathname] = uigetfile('NST08.xls');       %读入 8 家非 ST 公司数据
file=[pathname filename];
y1=xlsread(file);
ptest=[x1;y1];                                       %定义验证输入向量
ptest=ptest';
```

② 构造径向基概率神经网络。概率神经网络结构如图 17-12 所示，该网络为两层结构，输入层采用高斯径向基传递函数，输出层采用竞争传递函数，输出为一元向量，只有 1 和 2 两种取值，适用于模式分类问题。

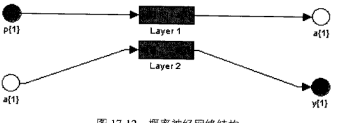


图 17-12 概率神经网络结构

在建立该网络模型时，使用的样本是上述 30 家 ST 公司和 30 家正常公司的财务数据。利用这 60 家公司的 26 个财务指标数据作为为径向基概率神经网络的学习训练样本，通过调整网络模型隐含层的节点数目，反复地学习和训练来确定网络的可调整参数，从而建立起径向基概率神经网络模型。输入下列命令，完成网络创建：

```
T2=T+1;
T2 = ind2vec(T2);
spread = 1;
net = newpnn(p,T2,spread);
```

函数 newpnn 生成概率神经网络，从而实现公司的分类。应用 newpnn 函数时，设定径向基函数的散布常数 spread=1.0。输出层输出只有两个值，即 Y=1 或者 Y=2。概率神经网络相对于 BP 神经网络而言，网络创建算法更为高效，生成速度更快，在构建之后不需要对网络进行训练便可直接应用。

③ 应用验证样本 ptest 对网络进行仿真，并绘图显示。输入命令：

```
A = sim(net,p);
Ac = vec2ind(A);
Atest = sim(net,ptest);
Actest = vec2ind(Atest)
plot(Actest,'r*')
```

绘出图形如图 17-13 所示。

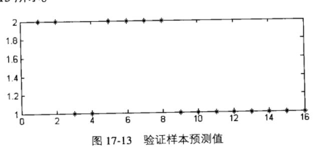


图 17-13 验证样本预测值

从图中可看出，被 ST 的 8 家公司中有两家被误判为正常公司，不过对 8 家正常公司的判断都正确。所以该模型是有效的，其预测准确率是非常理想的。

运用基于径向基的概率神经网络建立财务风险预测径向基概率神经网络模型，对验证样本预测的准确率能达到 87.5%，是一种比较理想的非统计方法的预测模型。

17.7 小结

本章首先概述了神经网络在预测中的应用，然后结合 5 个大例子，综合讲述运用 MATLAB 进行神经网络预测。



第 18 章 Simulink 中的神经网络设计

Simulink 是 MATLAB 环境下用于系统建模、动态分析的软件包。利用 Simulink，用户可以对所提出的问题构建相应系统，并且进行仿真，查看运行状况。

在 Simulink 环境下，用户可以通过简单的拖曳操作，或者根据需求修改现有的模块，很方便地构造出系统模型。Simulink 既支持线性系统，也支持非线性系统；既支持连续时间系统，也支持离散时间系统和混合系统，以及多抽样率系统。

在科学和工程领域，利用 Simulink 进行建模以及解决实际问题也是常见的方法。其应用领域包括航空航天与国防、工业自动化、通信、电子与信号处理、医疗器械等各个方面。

Simulink 提供 GUI 图形用户界面，用户可以通过绘制框图的方式实现建模过程。它还提供了各种信号源、线性与非线性组件、连接器模块库供用户选择。如果这些模块库不能够满足需求的话，用户可以创建自己的模块。利用 GUI 可以简化建模过程，迅速建立编程语言中需要用差分或微分方程才能够表示的模块。

在 Simulink 中各种模块是分级的，用户可以采用自顶向下或自底向上的方式创建模型，可以直接查看系统顶层框图，双击组件以查看其细节。此方式可以清晰地显示模型的组成结构和组件间连接的方式。

在建模过程完成之后，用户可以对其进行仿真，这可以直接通过 Simulink 菜单或 MATLAB 命令实现。以菜单方式进行仿真通常是很方便的，但是对于需要批处理的仿真过程，例如 Monte Carlo 模拟这样的过程，利用命令行方式则更为方便；而利用 Scope 模块或其他显示模块，可以方便地查看仿真结果。

此外，Simulink 可以完成模型分析的工作，这也可以通过 MATLAB 命令行的方式实现。

本章我们首先介绍 Simulink 的神经网络模块组件，然后通过实例介绍如何使用 Simulink 设计神经网络。

18.1 Simulink 神经网络模块

Simulink 中的神经网络模块组件（NN Toolbox Blockset）是 MATLAB 神经网络工具箱的重要组成部分，可以通过下面两种方式打开 Simulink 神经网络模块库。

（1）在 MATLAB 命令区中，输入命令 neural。

（2）单击“Simulink Library Browser”窗口左边子窗口中神经网络模块库前面的加号“+”。

利用 neural 命令打开的神经网络模块组件窗口如图 18-1 所示。

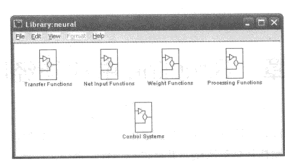


图 18-1　神经网络模块组件窗口

从图中可以看到，其中主要包括 5 类子模块组件，分别是：传递函数模块库（Transfer Functions）、网络输入函数模块库（Net Input Functions）、权值函数模块库（Weight Functions）、处理函数模块库（Processing Functions），以及控制系统模块库（Control Systems）。下面对这些子模块分别进行介绍。

18.1.1　传递函数模块库

双击图 18-1 所示的神经网络模块组件窗口中的 Transfer Functions 组件，将弹出传递函数模块窗口，如图 18-2 所示。

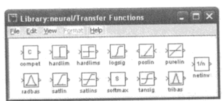


图 18-2　传递函数模块窗口

其中每一个方框代表一种网络传递函数，可供选择的传递函数包括以下几种。

- compet：竞争型传递函数。
- harlim：硬限值传递函数。
- hardlims：对称硬限值传递函数。
- logsig：对数 sigmoid 传递函数。
- poslin：正线性传递函数。
- purelin：纯线性传递函数。
- radbas：径向基传递函数。
- satlin：饱和线性传递函数。
- satlins：对称饱和线性传递函数。
- softmax：弱最大值传递函数。
- tansig：双曲正切 sigmoid 传递函数。
- tribas：三角基传递函数。

- netinv：反转型传递函数。

其中每一种传递函数的输入与输出的维度都相同。

18.1.2 网络输入函数模块库

双击图 18-1 所示的神经网络模块组件窗口中的“Net input Functions”组件，将弹出网络输入函数模块窗口，如图 18-3 所示。

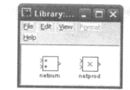


图 18-3 网络输入模块窗口

这个窗口包含两种网络输入函数。

- netsum：加减计算。
- netprod：向量点积运算。

其中每一种模块都能够计算任意维度的输入向量、加权层输出向量，以及偏差向量，返回一个网络输入向量。

18.1.3 权值函数模块库

双击图 18-1 所示的神经网络模块组件窗口中的“Weight Functions”组件，将弹出权值函数模块窗口，如图 18-4 所示。

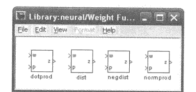


图 18-4 权值函数模块窗口

这个窗口包含 4 种权值函数。

- dotprod：点积权值函数。
- dist：欧氏距离权值函数。
- negdist：欧氏距离负值计算权值函数。
- normprod：归一化点积权值函数。

其中每一个模块都以一个神经元的权值向量作为输入，将其与输入向量进行计算，从而得到神经元的加权输入值。

需要指出的是，上述模块需要将神经元的权值向量定义为一个列向量，这是因为 Simulink 信号只能是列向量，而不能是矩阵或者行向量的形式。由于这个限制，如果用户

想要实现一个具有 S 个神经元的网络层的权值矩阵，就需要调用 S 个权值函数模块，每个模块代表一行。

18.1.4 处理函数模块库

双击图 18-1 所示的神经网络模块组件窗口中的“Processing Functions”组件，将弹出处理函数模块窗口，如图 18-5 所示。

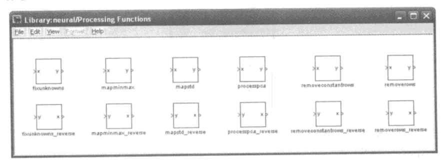


图 18-5 处理函数模块窗口

这个窗口包含 5 种处理函数及其对应的反函数，它们可以用来对输入进行预处理或者对输出进行后处理。

- fixunknowns：修正未知处理。
- mapminmax：映射最小最大处理。
- mapstd：标准差均值映射处理。
- processpca：procwsspca 处理。
- removeconstantrows：删除常数行处理。
- removerows：删除行处理。

18.1.5 控制系统模块库

双击图 18-1 所示的神经网络模块组件窗口中的“Control Systems”组件，将弹出控制系统模块窗口，如图 18-6 所示。

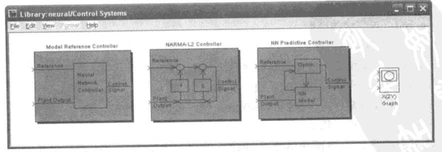


图 18-6 控制系统模块窗口

这个窗口包含了四个控制系统模块。

- Model Reference Controller：模型参考控制器。
- NARMA-L2 Controller：NARMA-L2 控制器。
- NN Predictive Controller：神经网络预测控制器。
- Graph：示波器模块。

前三种控制模块已经在前面介绍过了。最后的示波器模块在 Simulink 中可以完成实时显示的功能。

18.2 神经网络 Simulink 模型设计实例

本节我们将介绍神经网络 Simulink 模型设计实例。首先介绍相关的工具箱函数 gensim。此函数用于创建一个 Simulink 神经网络模块，以便应用于 Simulink 仿真。其调用格式如下：

gensim(net,st)

其中：

- net 为网络对象；
- st 指定了网络的采样时间（sample time），通常是一个正实数。

如果网络没有与输入权值或者神经元层权值相关的延迟，可将参数 st 设定为-1，这样函数 gensim 将生成一个连续采样的网络模型。

【例 18-1】 根据给定的输入和输出，设计一个线性神经网络，对其进行逼近，并利用 Simulink 建立系统模型进行仿真。

解： ① 定义输入与期望输出样本向量。在 MATLAB 工作区输入命令：

```
p = [1 2 3 4 5];
t = [1 3 5 7 9];
```

然后对此样本集应用 newlin 函数，生成一个线性神经网络。输入命令：

```
net = newlind(p,t)
```

下面对网络进行仿真检验。输入命令：

```
y = sim(net,p)
```

得到输出结果为：

```
y = 1.0000    3.0000    5.0000    7.0000    9.0000
```

仿真结果说明，网络能够很好地完成对输入、输出样本向量的线性逼近。

② 下面利用 gensim 函数生成上述网络的 Simulink 模型。输入命令：

```
gensim(net,-1)
```

其中设定第二个参数为-1，表示生成一个连续时间采样的网络模块。执行此命令，将弹出一个 Simulink 窗口，如图 18-7 所示。

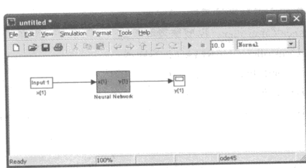


图 18-7　生成的网络模型窗口

图中包含了一个“Neural Network”模块，就是线性网络生成的 Simulink 仿真模型。其输入端连接一个信源输入，输出端连接一个示波器模块。双击“Neural Network”模块，可以查看其详细结构，弹出的图形如图 18-8 所示。

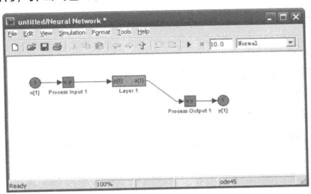


图 18-8　网络模型结构

可以看到，网络为单层结构，有一个输入端“$x\{1\}$”和一个输出端“$y\{1\}$”，“Layer 1”是神经元层，双击此“Layer 1”模块，可以进一步查看此神经网络层的详细结构，如图 18-9 所示。

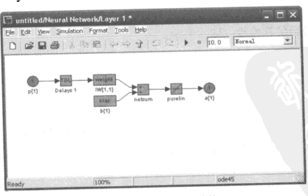


图 18-9　线性层内部结构

可以看到，此线性层中包含延迟线、网络权值、偏差、求和网络输入函数、线性传递函数等单元，进一步双击各个单元，可以得到各自的属性对话框，通过编辑对话框中的属性值，可以改变网络的属性。

③ 下面对此 Simulink 网络模块进行仿真。

双击图 18-7 窗口中的“Input1”输入模块，弹出其属性对话框，其中需要输入一个常数作为输入，此初始常数值为 0.089，是一个随机产生的常数，将其中的常数值改为 2，如图 18-10 所示。

单击“OK”按钮，回到 Simulink 工作区，单击工作窗口中的启动仿真按钮 ▶ 开始仿真，就可以在显示屏上看到输出波形，如图 18-11 所示。

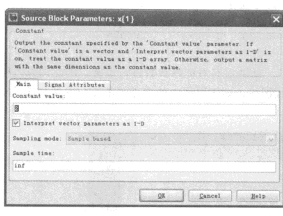


图 18-10 输入模块的属性对话框

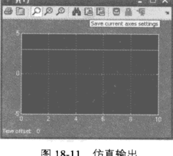


图 18-11 仿真输出

可以看到，输出值为 3，这表明模型输出结果是正确的。

对于 gensim 函数的应用和 Simulink 模块的仿真，下面我们再以一个输入为向量的例子来进行介绍。

【例 18-2】 给定一个多元输入向量和一维输出向量，设计前向 BP 神经网络对其关系进行拟合，并建立 Simulink 模型仿真。

解：① 定义输入与期望输出样本向量。在 MATLAB 工作区输入命令：

```
x=[0.1  0.5398  0.5325  0.5324;
   0.1 -0.9341  0.9339 -0.9327;
   0.1 -6.4617  0.8567  0.8850;
   0.1 10.3576 -10.1934 8.9586;
   0.1 10.9531 31.1317 51.2697];
t=[0.1 10.9630 31.1417 51.2796];
```

x 是原始的输入向量，为 5×4 矩阵，因此输入向量为五元列向量，t 是原始的输出向量，我们首先对其进行归一化处理。输入命令：

```
for i=1:5
    P(i,:)=x(i,:)/max(abs(x(i,:)));
end
```

```
P,
T=t/max(abs(t)),
```

输出结果为:

```
P = 0.1853    1.0000    0.9865    0.9863
    0.1071   -1.0000    0.9998   -0.9985
    0.0155   -1.0000    0.1326    0.1370
    0.0097    1.0000   -0.9841    0.8649
    0.0020    0.2136    0.6072    1.0000
T = 0.0020    0.2138    0.6073    1.0000
```

以上即归一化后的输入输出向量，下面我们利用此向量来构建 BP 网络。

② 建立网络，并对其进行训练。输入命令:

```
net=newff(minmax(P),[5,1],{'tansig','logsig'},'trainlm');
net=train(net,P,T);
```

训练过程中将弹出 nntraintool 窗口，此处没有对训练参数进行设置，因此训练过程采用默认设置，对网络进行 1000 次迭代训练，训练过程中的误差性能变化曲线如图 18-12 所示。

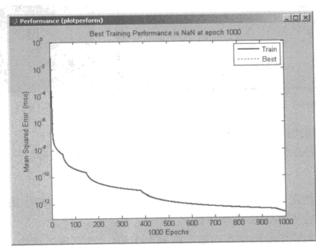


图 18-12 误差性能变化曲线

③ 利用 gensim 函数生成上述网络的 Simulink 模型。输入命令:

```
gensim(net,-1)
```

执行此命令，生成的是一个连续时间采样的网络模块，同时弹出 Simulink 窗口，如图 18-13 所示。

此网络就是训练完毕的网络，下面我们对其进行仿真。

④ 双击数据源模块，将变量设置为输入向量 ***P*** 的第二列向量，即[1;-1;-1;1;0.2136]，如图 18-14 所示。

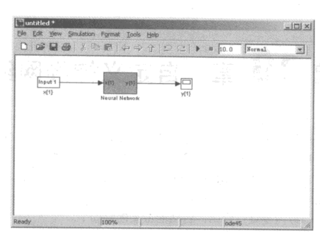


图 18-13　生成 BP 网络的 Simulink 模型

单击“OK”按钮回到 Simulink 工作区。双击显示模块打开显示窗口，然后单击启动仿真按钮 ▶ 开始仿真，就可以在显示屏上看到仿真结果，如图 18-15 所示。

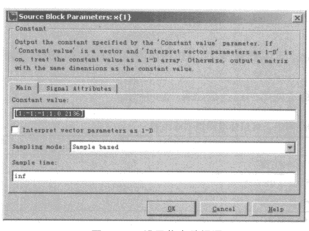


图 18-14　设置仿真数据源

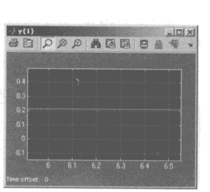


图 18-15　仿真结果

可以看到，确实如期望的那样，输出值为 0.21，等于输出向量元素 *T*(2)的值。

18.3 小结

本章讲述了利用 Simulink 进行神经网络设计的基本方法和实例，更复杂的例子读者可以结合神经网络控制一章进行学习，该章中三个控制系统实例都具有一定的复杂性，需要进一步深入的学习。

第 19 章　自定义神经网络

对于大多数应用情况来说，使用前面各章所介绍的 MATLAB 工具箱函数就能够设计出切合问题需要的神经网络。然而，对于一些特定的需求，单纯依靠工具箱提供的函数是不够的。为了解决此问题，MATLAB 工具箱提供了一种自定义网络结构对象。应用此对象，用户可以任意指定网络的类型，并且在建立之后对其进行初始化、训练和仿真等操作。

本章将对这一部分内容进行介绍，包括的主要内容如下。

（1）自定义网络。

（2）相关工具箱函数。

（3）自定义函数。

19.1 自定义网络

通过在工作区输入命令 nnnetwork，可以查看 MATLAB 神经网络工具箱提供的标准网络创建函数。

MATLAB 神经网络工具箱中的标准网络创建函数如表 19-1 所示。

表 19-1　神经网络工具箱标准网络创建函数

函　数	功　能
newc	创建一个竞争神经元层
newcf	创建一个前向级联 BP 网络
newdtdnn	创建一个分布延迟网络
newelm	创建一个 Elman 网络
newff	创建一个前向型 BP 网络
newfftd	创建一个前向型输入延迟 BP 网络
newfit	创建一个 Fitting 网络
newgrnn	创建一个广义回归神经网络
newhop	创建一个 Hopfield 反馈网络
newlin	创建一个线性网络层
newlind	设计一个线性网络
newlrn	创建一个分层反馈网络
newlvq	创建一个学习矢量量化网络
newnarx	创建一个有输出到输入反馈的 BP 网络
newp	创建一个感知器神经网络
newpnn	创建一个概率神经网络

续表

函　数	功　能
newpr	创建一个模式识别网络
newrb	创建一个径向基神经网络
newrbe	创建一个径向基网络
newsom	创建一个自组织映射网络

自定义神经网络设计的灵活性是通过面向对象的表示方式获得的。在以对象结构为基础的表示方式下，用户可以根据需要定义不同的网络结构，并对网络设置不同的算法，从而完成满足具体需要的设计。

我们可以从一个空网络开始自定义网络设计，这个空的网络对象可以利用函数 network 得到。这个新生成的结构体对象 net 中包含了许多待定的参数，这些都需要用户根据自身的需求自行定义。

下面我们通过一个实例来介绍自定义网络的过程。

19.1.1　定制网络

在开始创建自定义网络之前，首先需要确定它的结构。本节将介绍如何以自定义方式生成一个如图 19-1 所示的复杂网络。

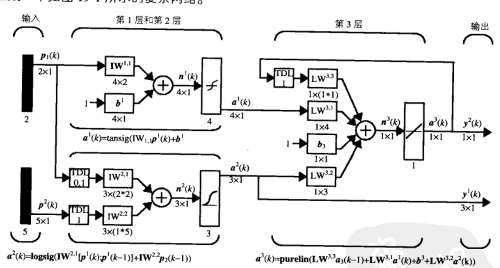


图 19-1　需要自定义生成的网络结构

图中的网络具有三个神经元层，分别为第 1 层、第 2 层和第 3 层。第 1 层采用 tansig 传递函数；第 2 层带有延迟线，采用 logsig 传递函数；第 3 层采用 purelin 传递函数。

网络有两个输入向量，分别为 $\boldsymbol{p}^1$(k)，$\boldsymbol{p}^2$(k)，输入向量 $\boldsymbol{p}^1$(k)为二元向量，输入取值范围为[0, 10]，向量 $\boldsymbol{p}^2$(k)为五元向量，取值范围为[−2, 2]。

在开始设计网络之前，首先需要确定网络的初始化方案和训练算法。在这里，每一层

都采用 Nguyen-Widrow 算法进行初始化，对应算法函数为 initnw。每一层的网络训练函数均采用 Levenberg-Marquartdt 反向传播算法（对应算法函数为 trainlm）。

给定输入向量样本，训练中采用第 3 层输出的均方误差值作为网络的误差性能函数。

19.1.2 定义网络

本节将通过下面的实例，一步步讲述如何自定义神经网络。

【例 19-1】 利用自定义方式生成如图 19-1 所示的神经网络。

解： ① 生成空网络，首先通过 network 函数创建一个新网络，并查看其属性值。输入命令：

```
net=network
```

便生成了神经网络。在生成的网络中，各种属性值均为空值，即生成的网络是一个空的神经网络对象。

② 修改网络属性值，对网络进行自定义。

首先将网络的输入数目设定为 2 个，网络层数设定为 3 层。输入如下命令：

```
net.numInputs = 2;
net.numLayers = 3;
```

需要指出的是，numInputs 属性指定的是输入向量源的个数，而不是输入向量中的元素的个数，向量中元素的个数由参数 net.inputs{i}.size 给出。

完成上述定义之后，输入 net 命令查看网络属性的变化情况。输入命令：

```
net
```

可以看到部分输出结果为：

```
net =    Neural Network object:
    architecture:
         numInputs: 2
         numLayers: 3
         biasConnect: [0; 0; 0]
         inputConnect: [0 0; 0 0; 0 0]
         layerConnect: [0 0 0; 0 0 0; 0 0 0]
         outputConnect: [0 0 0]
```

可以看到，网络输入向量数和层数都变成了修改后的值。而偏差连接、输入连接、层连接、输出连接属性都是零向量。这表示网络输入、输出，以及各层之间没有任何的连接。

③ 调整网络的偏差连接属性。

网络的偏差连接矩阵是一个 3×1 的向量，为创建一个第 i 层的偏差连接需要将 net.biasConnect{i}的值设为 1。下面将指定网络的第 1 层与第 3 层有偏差连接，如图 19-1 所绘出的那样。输入如下命令：

```
net.biasConnect(1) = 1;
net.biasConnect(3) = 1;
```

或采用单行的方式，输入命令：

```
net.biasConnect = [1; 0; 1];
```

就可以完成对网络偏差的指定。

④ 指定输入连接、层权值连接属性。

输入连接矩阵是一个 3×2 的矩阵，表示有两个输入向量（信号源），而有三个目标层（接收方）。因此，参数 net.inputConnect(i,j)表征了第 *j* 个输入到第 *i* 层的连接情况。

为了在第 1 个输入与第 1 层、第 2 层之间建立连接，以及在第 2 个输入与第 2 层建立连接，输入如下命令：

```
net.inputConnect(1,1) = 1;
net.inputConnect(2,1) = 1;
net.inputConnect(2,2) = 1;
```

或者采用单行的方式，输入命令：

```
net.inputConnect = [1 0; 1 1; 0 0];
```

即可完成输入连接属性的设置。

类似的，net.layerConnect(i,j)表征了从第 *j* 层到第 *i* 层的权值连接情况。要将第 1、2、3 层与第 3 层连接起来，输入如下命令：

```
net.layerConnect(3,1) = 1;
net.layerConnect(3,2) = 1;
net.layerConnect(3,3) = 1;
```

或以单行的方式，输入命令：

```
net.layerConnect = [0 0 0; 0 0 0; 1 1 1];
```

即可完成权值连接的设置。

⑤ 指定输出连接属性。

输出连接矩阵是一个 1×3 的向量，表示具有 3 个输出源（三个网络层），1 个输出目标（网络外部），如果网络层到输出目标无连接，对应值设为 0，有连接则设置为 1。

从图 19-1 中可看到，网络的第 3 层与输出有连接。输入如下命令：

```
net.outputConnect = [0 0 1];
```

即表示第 3 层与输出有连接，如此便完成了输出连接的设置。

完成上述网络结构属性的设置之后，可以通过输入 net 命令来查看网络的属性值。更改后的网络属性如下：

```
net =Neural Network object:
   architecture:
        numInputs: 2
        numLayers: 3
        biasConnect: [1; 0; 1]
        inputConnect: [1 0; 1 1; 0 0]
        layerConnect: [0 0 0; 0 0 0; 1 1 1]
        outputConnect: [0 0 1]
        numOutputs: 1  (read-only)
```

```
        numInputDelays: 1  (read-only)
        numLayerDelays: 1  (read-only)
```

而网络的子结构属性为：

```
    subobject structures:
         inputs: {2x1 cell} of inputs
         layers: {3x1 cell} of layers
        outputs: {1x3 cell} containing 1 output
         biases: {3x1 cell} containing 2 biases
   inputWeights: {3x2 cell} containing 3 input weights
   layerWeights: {3x3 cell} containing 3 layer weights
```

可见给定的网络有两个输出，分别来自第 2 层与第 3 层。

⑥ 设定输入属性。

当我们把输入数目 net.numInputs 的值设定为 2 时，输入属性就成为了一个包含两个输入结构的 cell 数组。其中，作为组元的第 *i* 个输入结构体表示为 net.inputs{i}，包含了与第 *i* 个输入的相关信息。

为查看输入属性结构体，可以输入如下命令：

```
net.inputs
```

输出结果为：

```
ans = [1x1 struct]
      [1x1 struct]
```

继续查看第 1 个输入结构体的内容。输入命令：

```
net.inputs{1}
```

输出结果为：

```
ans =
    exampleInput: [0 1]
    processFcns: {}
    processParams: {}
    processSettings: {}
    processedRange: [0 1]
    processedSize: 1
    range: [0 1]
    size: 1
    userdata: [1x1 struct]
```

我们可以设定样本输入（exampleInput）的值，那么其后的参数就会自动的根据样本输入的值设定。

通过如下命令对样本输入进行设定：

```
net.inputs{1}.exampleInput = [0 10 5; 0 3 10];
```

此后，重新查看第 1 个输入结构体的值，其结果如下所示：

```
ans =
```

```
        exampleInput: [2x3 double]
        name: 'Input'
        processFcns: {'removeconstantrows'  'mapminmax'}
        processParams: {[1x1 struct]  [1x1 struct]}
        processSettings: {[1x1 struct]  [1x1 struct]}
        processedRange: [2x2 double]
        processedSize: 2
        range: [2x2 double]
        size: 2
        userdata: [1x1 struct]
```

可见，根据我们所指定的样本值，exampleInput 变成了一个 2×3 的矩阵，而且 name、processFcns 等属性值确实根据样本输入值自动进行了调整。其中处理函数 processFcns 的值为{'removeconstantrows' 'mapminmax'}，表示在网络进行训练或仿真之前，首先要删除输入的常数行，并且对输入的最大和最小值进行映射，随后的参数 processParams 指定的是处理函数的参数。

下面在不指定处理函数的情况下，将第 2 个输入的大小设定为 5。输入如下命令：

```
net.inputs{2}.size = 5;
```

⑦ 设定层属性。

当我们将层数目参数 net.numLayers 指定为 3 时，层属性就成为了一个 3 维的 cell 数组。可以输入下面的命令，查看第 1 层的属性：

```
net.layers{1}
```

得到的结果为：

```
ans =
    dimensions: 1
    distanceFcn: 'dist'
    distances: 0
    initFcn: 'initwb'
    netInputFcn: 'netsum'
    netInputParam: [1x1 struct]
    positions: 0
    size: 1
    topologyFcn: 'hextop'
    transferFcn: 'purelin'
    transferParam: [1x1 struct]
    userdata: [1x1 struct]
```

现在需要将第 1 层设置为 4 个神经元，传递函数设置为 tansig，初始化函数设定为 Nguyen-Widrow 函数。输入命令：

```
net.layers{1}.size = 4;
net.layers{1}.transferFcn = 'tansig';
net.layers{1}.initFcn = 'initnw';
```

将第 2 层设置为 3 个神经元，传递函数设置为 logsig，初始化函数同样设定为 Nguyen-Widrow 函数。输入命令：

```
net.layers{2}.size = 3;
net.layers{2}.transferFcn = 'logsig';
net.layers{2}.initFcn = 'initnw';
```

对于第 3 层，其大小默认为 1 个神经元，传递函数默认为 purelin，这正符合我们想要构建的网络的需求，因此不需要更改，但是我们还需要指定网络层的初始化函数。输入命令：

```
net.layers{3}.initFcn = 'initnw';
```

以上就完成了网络层属性的设定。

⑧ 设定输出向量的属性。

首先，可以查看目前输出向量的属性。在 MATLAB 工作区输入如下命令：

```
net.outputs
```

输出结果为：

```
ans = []    []    [1x1 struct]
```

可以看到，输出属性为一个 1×3 的 cell 数组，其中第 3 个 cell 单元为结构体类型，代表第 3 层的输出。这是在我们将参数 net.outputConnect 设定为[0 0 1]的时候自动确定的。

查看第 2 层的输出结构体，输入命令：

```
net.outputs{2}
```

输出结果为：

```
ans =
    exampleOutput: [3x2 double]
    name: 'Output'
    processFcns: {}
    processParams: {}
    processSettings: {}
    processedRange: [3x2 double]
    processedSize: 3
    range: 3
    size: 3
    userdata: [1x1 struct]
```

可以看到，size 的值为 3，这是在我们将 net.layer{2}.size 设定为 3 时就自动设定的。也可以通过类似的方法检查第 3 层的输出情况。输入命令：

```
net.outputs{3}
```

输出结果为：

```
ans =
    exampleOutput: [-Inf Inf]
    name: 'Output'
    processFcns: {}
    processParams: {}
    processSettings: {}
    processedRange: [-Inf Inf]
```

```
    processedSize: 1
    range: 1
    size: 1
    userdata: [1x1 struct]
```

可见第 3 层输出的 size 为 1，这是期望的值。由于第 1 层没有输出，因此如果用类似的方法查看将无效，得到的输出结果是一个空集。

需要指出的是，在较低版本的工具箱中，输出与目标响应两者是分开的，因此还需要指定目标响应的属性。但是在高版本神经网络工具箱中，与处理过程相关的各种属性是自动应用在训练过程中的目标响应上的，因此不需要单独指定目标响应的属性。

与输入处理属性类似，如果对 exampleOutput 属性进行了设定，则其后的参数都会自动更新。

⑨ 设定偏差、输入权值、层权值属性。

首先查看偏差属性。在 MATLAB 工作区输入如下命令：

```
net.biases
```

输出结果为：

```
ans =
    [1x1 struct]
    []
    [1x1 struct]
```

以上显示的就是偏差属性 cell 数组。然后查看输入权值属性，输入命令：

```
net.inputWeights
```

输出结果为：

```
ans = [1x1 struct]       []
    [1x1 struct]    [1x1 struct]
        []               []
```

这是输入权值属性 cell 数组。接下来查看层权值属性，输入命令：

```
net.layerWeights
```

输出结果为：

```
ans =       []                []                []
            []                []                []
        [1x1 struct]     [1x1 struct]     [1x1 struct]
```

其中各个矩阵中包含结构体的位置就是相应的连接值 net.biasConnect 为 1 的位置。还可以进一步查看各个结构体的值，例如查看偏差向量中第 1 个结构体的值。输入如下命令：

```
net.biases{1}
```

输出结果为：

```
ans =
    initFcn: ''
```

```
    learn: 1
    learnFcn: ''
    learnParam: ''
    size: 4
    userdata: [1x1 struct]
```

类似的，输入下面的命令，还可以查看各个具体结构体的值。

```
net.biases{3}
net.inputWeights{1,1}
net.inputWeights{2,1}
net.inputWeights{2,2}
net.layerWeights{3,1}
net.layerWeights{3,2}
net.layerWeights{3,3}
```

这里不再一一列出以上命令的执行结果，读者可执行命令自行查看。下面根据图 19-1 中网络的结构，对权值中的延迟线进行指定。输入命令：

```
net.inputWeights{2,1}.delays = [0 1];
net.inputWeights{2,2}.delays = 1;
net.layerWeights{3,3}.delays = 1;
```

以上输入表明，第 1 个输入到网络第 2 层的延迟线结构为[0 1]，第 2 个输入到网络第 2 层的延迟为 1，从第 3 层输出到第 3 层输入的延迟为 1。

⑩ 设定网络函数。

从网络属性中我们可以看到，网络中各种函数都是没有设置的，显示如下：

```
    functions:
        adaptFcn: (none)
        divideFcn: (none)
        gradientFcn: (none)
        initFcn: (none)
        performFcn: (none)
        plotFcns: {}
        trainFcn: (none)
```

将网络初始化函数 initFcn 设置为 initlay，表示网络将采用层的初始化函数完成网络自身的初始化，也就是采用 Nguyen-Widrow 函数进行初始化。输入命令：

```
net.initFcn = 'initlay';
```

将网络性能函数 perfermFcn 设置为 mse，也就是均方误差。将训练函数 trainFcn 设置为 trainlm，也就是 Levenberg-Marquardt 反向传播算法，这是符合图 19-1 网络要求的设置。在 MATLAB 工作区中输入命令：

```
net.performFcn = 'mse';
net.trainFcn = 'trainlm';
```

将数据划分函数 divideFcn 设置为 dividerand，即随机划分训练数据。输入命令：

```
net.divideFcn = 'dividerand';
```

在有监督训练的情况下，输入和目标响应数据以随机的方式划分为数个训练、测试、验证数据集。在学习过程中，通过训练数据集对网络进行训练，直到通过验证数据集验证到其网络性能开始下降为止，此时性能达到峰值。接下来，利用测试数据集对网络性能提供另外一次完全独立的测试。

将绘图函数 plotFcn 设置为 plotperform 函数和 plottrainstate 函数。其中，plotperform 对训练、验证、测试过程中的性能函数绘图显示，而 plottrainstate 函数则对训练算法中每一阶段的状态绘图显示。输入命令：

```
net.plotFcns = {'plotperform','plottrainstate'};
```

以上就完成了网络函数的设置。输入 net 命令检查设置后的网络函数，结果为：

```
functions:
    adaptFcn: (none)
    divideFcn: 'dividerand'
    gradientFcn: 'gdefaults'
    initFcn: 'initlay'
    performFcn: 'mse'
    plotFcns: {'plotperform','plottrainstate'}
    trainFcn: 'trainlm'
```

⑪ 设定权值和偏差值。

在此之前，再查看一下网络属性中最后一组的情况，其值为：

```
IW: {3x2 cell} containing 3 input weight matrices
LW: {3x3 cell} containing 3 layer weight matrices
b: {3x1 cell} containing 2 bias vectors
```

其中的 cell 数组包含连接属性(即 net.inputConnect，net.layerConnect，net.biasConnect)、也包含子对象属性（ net.inputWeights，net.layerWeights，net.biases ），以及网络权值矩阵和偏差向量。

输入下列命令，可以查看各个权值和偏差向量的值。

```
net.IW{1,1}, net.IW{2,1}, net.IW{2,2}
net.LW{3,1}, net.LW{3,2}, net.LW{3,3}
net.b{1}, net.b{3}
```

例如，查看第 1 层输入权值矩阵 IW{1，1}的值。输入命令：

```
net.IW{1,1}
```

输出结果为：

```
ans = 0     0
      0     0
      0     0
      0     0
```

第 1 层输入权值矩阵 IW{1，1}的行数与第 1 层神经元数目（ net.layers{1}.size ）相同，而第 1 层输入权值矩阵的列数与网络输入的维数（ net.inputs{1}.size ）相等。由上面的执行

结果可见，第 1 层的输入权值矩阵元素的值均设置为 0。

同样，输入命令：

```
net.LW{3,1}
```

输出结果为：

```
ans =     0     0     0     0
```

层权值矩阵 net.LW{3,1}的列数等于第 1 层神经元数目（net.layers{1}.size）与延迟值（length(net.inputWeights{i,j}.delays)的乘积。上述结果表明第 1 层连接权值初值均为 0。

输入命令：

```
net.b{1}
```

输出结果为：

```
ans = 0
      0
      0
      0
```

这表明第 1 层偏差初值均设置为 0。对于此例，我们保持其初始值为 0 不变即可。

19.1.3 网络行为

在 19.1.2 节完成了网络结构的属性设置，下面就可以开始对网络的权值偏差进行设置、训练，从而完成仿真。本节延续上面的步骤，继续进行自定义。

⑫ 网络初始化。输入如下命令：

```
net = init(net);
```

通过下列命令，可以再次检查网络的权值和偏差矩阵。

```
net.IW{1,1}, net.IW{2,1}, net.IW{2,2}
net.LW{3,1}, net.LW{3,2}, net.LW{3,3}
net.b{1}, net.b{3}
```

例如，输入命令：

```
net.IW{1,1},
```

输出结果为：

```
ans =
    2.5810    1.0855
    1.9880   -1.9717
   -2.4075   -1.4296
    2.7822    0.3155
```

输入命令：

```
net.LW{3,1},
```

输出结果为：

```
ans =   -0.6576    0.4121   -0.9363   -0.4462
```

输入命令：

```
net.b{1},
```

输出结果为：

```
ans =
   -2.8000
   -0.9333
   -0.9333
    2.8000
```

以上即完成了网络权值和偏差的初始化。

⑬ 对网络进行训练，定义一个具有两个输入向量的 cell 数组作为样本输入，一个为二元向量，另一个为五元向量。输入命令：

```
P = {[0; 0] [2; 0.5]; [2; -2; 1; 0; 1] [-1; -1; 1; 0; 1]},
```

输出结果为：

```
P =
    [2x1 double]    [2x1 double]
    [5x1 double]    [5x1 double]
```

同时定义期望目标响应，也是 cell 数组，具有两个元素。输入命令：

```
T = {1 -1},
```

输出结果为：

```
T = [1]    [-1]
```

在开始训练之前，可以先利用输入样本对初始化的网络进行仿真，查看它的输出是否与期望目标响应接近。输入命令：

```
Y = sim(net,P)
```

输出结果为：

```
Y = [    1.4709]    [    1.1346]
```

输出 ***Y*** 正是对应输入向量 ***P*** 的两组输出值。从上面的结果可以看到，第 2 组输出与期望响应–1 相差是非常远的，初始化的网络还不能够完成对应的功能，因此需要对网络进行训练。

首先查看训练参数。输入如下命令：

```
net.trainParam
```

输出结果为：

```
ans =
```

```
  show: 25
  showWindow: 1
  showCommandLine: 0
  epochs: 1000
  time: Inf
  goal: 0
  max_fail: 6
  mem_reduc: 1
  min_grad: 1.0000e-010
  mu: 1.0000e-003
  mu_dec: 0.1000
  mu_inc: 10
  mu_max: 1.0000e+010
```

其中训练目标值初始设定值 goal 为 0，我们将其修改为 1e–10。输入命令：

```
net.trainParam.goal = 1e-10;
```

然后调用 train 函数开始训练。输入命令：

```
net = train(net,P,T);
```

训练完毕，三步之后就达到了训练目标。网络误差性能的变化如图 19-2 所示。

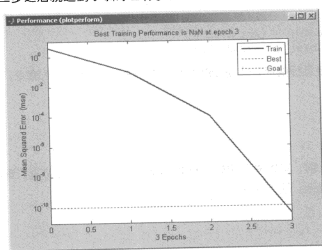


图 19-2　训练过程中网络误差性能的变化

对训练后的网络进行仿真。输入命令：

```
Y = sim(net,P)
```

得到结果为：

```
Y =     [1.0000]    [-1.0000]
```

可见训练后的网络实现了预期的功能。以上我们就介绍了以自定义的方式实现网络的全过程。用户在处理实际问题时，可以用类似的方法灵活实现网络的自定义。

19.2 相关工具箱函数

前面的章节中介绍的神经网络工具箱函数都是针对标准工具箱网络设定的，本节将介绍的是与自定义网络相关的工具箱函数，这些函数可以用于用户自定义网络，主要包括初始化函数、传递函数以及学习函数几个部分。

19.2.1 初始化函数

自定义网络相关的初始化函数有 randnc 和 randnr，下面对其进行介绍。

1．randnc

此函数是一个权值初始化函数，其调用格式为：

W = randnc(**S**,PR)

W = randnc(**S**,**R**)

其中：

- *S* 为神经元的数目；
- PR 为 *R*×2 的矩阵，取值范围为[Pmin Pmax]。

函数返回一个随机的初始化权值矩阵，并且矩阵列向量已经归一化。

2．randnr

此函数也是权值初始化函数，其调用格式为：

W = randnr(**S**,PR)

W = randnr(**S**,**R**)

与 randnc 类似，其中 *S* 也是神经元的输入；PR 为 *R*×2 的矩阵，代表输入取值范围。函数执行后同样也是返回一个初始化的权值矩阵，但矩阵采用的是行向量归一化。

19.2.2 传递函数

自定义网络相关的传递函数包括 satlin、softmax 和 tribas，下面逐一介绍。

1．satlin

此函数为饱和线性传递函数。其调用格式为：

A = satlin(**N**, FP)

其中：

- **N** 为 *S*×*Q* 的矩阵，代表网络输入列向量；
- FP 是函数参数结构体，可以省略。

此传递函数仍然是一个线性函数，但与 purelin 不同的是，其取值范围限定在[0,1]内，其最小值为 0，最大值为 1。

利用下面代码可以得到 satlin 函数的图：

```
n = -2:0.1:2;
a = satlin(n);
plot(n,a)
```

绘出的图形如图 19-3 所示。

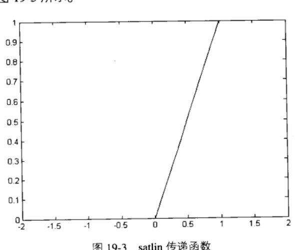


图 19-3　satlin 传递函数

此函数应用于网络时，可以采用下面的语句：

```
net.layers{i}.transferFcn = 'satlin';
```

即将第 i 层的传递函数设置成为 satlin 函数。

2. softmax

此传递函数是竞争传递函数的一个软版本，具有最大网络输入的神经元得到一个比较接近于 1 的输出，而其他神经元则得到比较接近于 0 的输出。这与竞争函数输出为绝对的 1 和 0 是有所不同的。其调用格式为：

$$A=\text{softmax}(N)$$

其中 N 为 $S\times Q$ 的网络输入向量。

我们可以利用下面的代码来观察 softmax 函数的性能。输入如下命令：

```
n = [0; 1; -0.5; 0.5];
a = softmax(n);
subplot(2,1,1), bar(n), ylabel('n')
subplot(2,1,2), bar(a), ylabel('a')
```

绘图的结果如图 19-4 所示。

在网络中调用此函数时通常用如下语句：

```
net.layers{i}.transferFcn = 'softmax';
```

即将第 i 层的传递函数设置成为 softmax 函数。

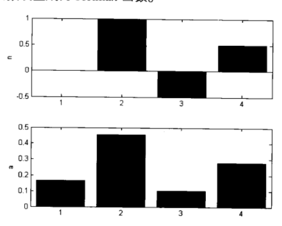


图 19-4 softmax 传递函数

3. tribas

此传递函数是一个三角形的径向基函数，其调用格式为：

```
A = tribas(N,FP)
dA_dN = tribas('dn',N,A,FP)
info = tribas(code)
```

其中 N 为 $S\times Q$ 的网络输入向量。

通过下面的代码可以绘制出三角径向基函数的图形。输入命令：

```
n = -2:0.1:2;
a = tribas(n);
plot(n,a)
```

绘出的图形如图 19-5 所示。

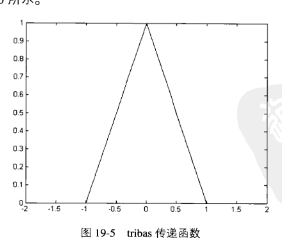


图 19-5 tribas 传递函数

此函数应用于网络中，可以采用下面的语句：

```
net.layers{i}.transferFcn = 'tribas';
```

即将第 i 层的传递函数设置成为 tribas 函数。

19.2.3 学习函数

神经网络工具箱中非标准的学习函数还包括 learnh、learnhd、learnis 以及 learnos，下面分别对其进行介绍。

1. learnh

此函数为 Hebb 权值学习函数，按照乘积、权值输入、神经元输出的比例来改变权值。其调用格式为：

[dW,LS] = learnh(***W***, ***P***, ***Z***, ***N***, ***A***, ***T***, ***E***, gW, gA, D, LP, LS)

info = learnh(code)

其中：

- ***W*** 为 $S\times R$ 的权值矩阵；
- ***P*** 为 $R\times Q$ 的输入向量；
- ***Z*** 为 $S\times Q$ 的加权输入向量；
- ***N*** 为 $S\times Q$ 的网络输入向量；
- ***A*** 为 $S\times Q$ 的输出向量；
- ***T*** 为 $S\times Q$ 的层目标向量；
- ***E*** 为 $S\times Q$ 的层误差向量；
- gW 为 $S\times R$ 的性能相关梯度；
- gA 为 $S\times Q$ 的性能相关输出梯度；
- *D* 为 $S\times S$ 的神经元距离；
- LP 为学习参数；
- LS 为学习状态，初始值为[]。

函数返回值为：

- dW 为 $S\times R$ 的权值调整矩阵；
- LS 为新的学习状态。

此函数执行过程中，学习速率的默认值：

```
LP.lr = 0.01
```

下面举例说明此函数的应用。应用下面的代码，建立一个随机输入向量 ***P*** 和输出向量 ***A***，同时定义学习速率。输入命令：

```
p = rand(2,1);
a = rand(3,1);
lp.lr = 0.5;
```

learnh 函数只需要上面的几个参数值即可进行权值调整的计算，因此可以应用下面的语句来计算权值调整的值。

```
dW = learnh([],p,[],[],a,[],[],[],[],[],lp,[])
```

得到的计算结果为：

```
dW =
    0.1599    0.1040
    0.0862    0.0561
    0.1820    0.1183
```

此结果就是应用函数 learnh 进行学习所得到的权值的调整量。

在网络中应用函数 learnh 的步骤如下。

① 设置 net.trainFcn 为 trainr。

② 设置 net.adaptFcn 为 trains。

③ 将每个 net.inputWeights{i, j}.learnFcn 的值都设为 learnh，将每个 net.layerWeights{i, j}.learnFcn 的值都设为 learnh。

④ 对网络进行训练。将 net.trainParam 的属性设置为需要的值。

⑤ 调用 train 函数。

以上就是函数 learnh 在网络训练中应用的步骤。

2. learnhd

此函数为退化的 Hebb 权值学习函数，它的学习速率随着时间步长递减。其调用格式如下：

[dW,LS] = learnhd(***W***, ***P***, ***Z***, ***N***, ***A***, ***T***, ***E***, gW, gA, D, LP, LS)

info = learnhd(code)

其中：

- ***W*** 为 $S \times R$ 的权值矩阵；
- ***P*** 为 $R \times Q$ 的输入向量；
- ***Z*** 为 $S \times Q$ 的加权输入向量；
- ***N*** 为 $S \times Q$ 的网络输入向量；
- ***A*** 为 $S \times Q$ 的输出向量；
- ***T*** 为 $S \times Q$ 的层目标向量；
- ***E*** 为 $S \times Q$ 的层误差向量；
- gW 为 $S \times R$ 的性能相关梯度；
- gA 为 $S \times Q$ 的性能相关输出梯度；
- *D* 为 $S \times S$ 的神经元距离；
- LP 为学习参数；
- LS 为学习状态，初始值为[]。

函数返回值为：



- dW 为 $S\times R$ 的权值调整矩阵；
- LS 为新的学习状态。

此函数执行过程中，学习速率的默认值：

```
LP.lr=0.01
```

退化速率：

```
LP.dr=0.01
```

下面对举例说明函数 learnhd 的应用。应用下面的代码建立一个随机输入向量 ***P*** 和输出向量 ***A***，同时定义学习速率和退化速率。输入命令：

```
p = rand(2,1);
a = rand(3,1);
w = rand(3,2);
lp.dr = 0.05;
lp.lr = 0.5;
```

learnhd 只需要上面的几个参数值就可以进行权值调整的计算，因此可以应用下面的语句来计算权值调整的值。

```
dW = learnhd(w,p,[],[],a,[],[],[],[],[],lp,[])
```

得到的计算结果为：

```
dW =
   -0.0158   -0.0235
   -0.0151   -0.0132
    0.0718    0.0971
```

此结果就是应用函数 learnhd 进行学习所得到的权值的调整量。

在网络中应用函数 learnhd 的步骤如下。

① 设置 net.trainFcn 为 trainr。

② 设置 net.adaptFcn 为 trains。

③ 将每个 net.inputWeights{i, j}.learnFcn 的值都设为 learnhd，将每个 net.layerWeights{i, j}.learnFcn 的值都设为 learnhd。

④ 对网络进行训练。将 net.trainParam 的属性设置为需要的值。

⑤ 调用 train 函数。

以上就是在网络训练中应用函数 learnhd 的步骤。

3. learnis

此函数为内星权值学习函数，在此学习规则下，权值矩阵朝着输入向量的方向变动。其调用格式如下：

[dW,LS] = learnis(***W***, ***P***, ***Z***, ***N***, ***A***, ***T***, ***E***, gW, gA, D, LP, LS)

info = learnis(code)

其中：

- $\boldsymbol{W}$ 为 $S \times R$ 的权值矩阵；
- $\boldsymbol{P}$ 为 $R \times Q$ 的输入向量；
- $\boldsymbol{Z}$ 为 $S \times Q$ 的加权输入向量；
- $\boldsymbol{N}$ 为 $S \times Q$ 的网络输入向量；
- $\boldsymbol{A}$ 为 $S \times Q$ 的输出向量；
- $\boldsymbol{T}$ 为 $S \times Q$ 的层目标向量；
- $\boldsymbol{E}$ 为 $S \times Q$ 的层误差向量；
- gW 为 $S \times R$ 的性能相关梯度；
- gA 为 $S \times Q$ 的性能相关输出梯度；
- $\boldsymbol{D}$ 为 $S \times S$ 的神经元距离；
- LP 为学习参数；
- LS 为学习状态，初始值为[]。

函数返回值为：

- dW 为 $S \times R$ 的权值调整矩阵；
- LS 为新的学习状态。

此函数的执行过程中，学习速率的默认值：

```
LP.lr=0.01
```

下面是一个函数 learnis 调用的简单例子。应用下面的代码建立一个随机输入向量 $\boldsymbol{P}$ 和输出向量 $\boldsymbol{A}$，同时定义学习速率。

```
p = rand(2,1);
a = rand(3,1);
w = rand(3,2);
lp.lr = 0.5;
```

应用下面的语句来计算权值调整的值：

```
dW = learnis(w,p,[],[],a,[],[],[],[],[],lp,[])
```

得到的计算结果为：

```
dW =
    0.0125    0.0134
    0.0328   -0.0601
    0.0059   -0.0335
```

此结果就是应用函数 learnis 进行学习所得到的权值的调整量。

在网络中应用函数 learnis 的步骤如下。

① 设置 net.trainFcn 为 trainr。

② 设置 net.adaptFcn 为 trains。

③ 将每个 net.inputWeights{i, j}.learnFcn 的值都设为 learnis，将每个 net.layerWeights{i, j}.learnFcn 的值都设为 learnis。

④ 对网络进行训练。将 net.trainParam 的属性设置为需要的值。

⑤ 调用 train 函数。

以上就是在网络训练中应用函数 learnis 的步骤。

4. learnos

此函数为外星权值学习函数，在此学习规则下，权值矩阵从输入向量的方向朝着输出向量的方向变动。其调用格式如下：

[dW,LS] = learnos(***W***, ***P***, ***Z***, ***N***, ***A***, ***T***, ***E***, gW, gA, D, LP, LS)

info = learnos(code)

其中：

- ***W*** 为 $S \times R$ 的权值矩阵；
- ***P*** 为 $R \times Q$ 的输入向量；
- ***Z*** 为 $S \times Q$ 的加权输入向量；
- ***N*** 为 $S \times Q$ 的网络输入向量；
- ***A*** 为 $S \times Q$ 的输出向量；
- ***T*** 为 $S \times Q$ 的层目标向量；
- ***E*** 为 $S \times Q$ 的层误差向量；
- gW 为 $S \times R$ 的性能相关梯度；
- gA 为 $S \times Q$ 的性能相关输出梯度；
- *D* 为 $S \times S$ 的神经元距离；
- LP 为学习参数；
- LS 为学习状态，初始值为[]。

函数返回值为：

- dW 为 $S \times R$ 的权值调整矩阵；
- LS 为新的学习状态；

此函数执行过程中，学习速率的默认值：

```
LP.lr=0.01
```

下面是一个函数 learnos 调用的简单例子。应用下面的代码建立一个随机输入向量 ***P*** 和输出向量 ***A***，同时定义学习速率。

```
p = rand(2,1);
a = rand(3,1);
w = rand(3,2);
lp.lr = 0.5;
```

应用下面的语句来计算权值调整的值：

```
dW = learnos(w,p,[],[],a,[],[],[],[],[],lp,[])
```

得到的计算结果为：

```
dW =
    0.0556    0.0235
```

```
  0.0294    0.1214
 -0.0050    0.0667
```

此结果就是应用函数 learnos 进行学习所得到的权值的调整量。

在网络中应用函数 learnos 的步骤如下。

① 设置 net.trainFcn 为 trainr。

② 设置 net.adaptFcn 为 trains。

③ 将每个 net.inputWeights{i, j}.learnFcn 的值都设为 learnos，将每个 net.layerWeights{i, j}.learnFcn 的值都设为 learnos。

④ 对网络进行训练。将 net.trainParam 的属性设置为需要的值。

⑤ 调用 train 函数。

以上就是在网络训练中应用函数 learnos 的步骤。

19.3 自定义函数

利用 MATLAB 工具箱，可以自定义生成很多种类的函数，这为网络的仿真、训练等带来很大的方便性和灵活性。本节将对这些自定义函数进行详细的介绍。

根据其功能特点，自定义函数通常可以分为网络构建函数、初始化函数、学习函数以及自组织映射函数四大类，它们分别又包括各自的子类。

（1）网络构建函数

- 传递函数。
- 网络输入函数。
- 权值函数。

（2）初始化函数

- 网络初始化函数。
- 层初始化函数。
- 权值与偏差初始化函数。

（3）学习函数

- 网络训练函数。
- 自适应学习函数。
- 性能函数。
- 权值和偏差学习函数。

（4）自组织映射函数

- 拓扑函数。
- 距离函数。

下面各节对上述各种函数逐一进行介绍。



19.3.1 网络构建函数

网络构建函数包括传递函数、网络输入函数、权值函数。

1．传递函数

传递函数是根据给定的网络输入向量计算输出向量的函数，网络输入向量与输出向量必须具有相同的维数。

传递函数经过定义之后，可以应用在任意的网络层上。例如自定义的传递函数为 tamplate_transfer，将其应用在网络的第 i 层上，可以通过下面的语句实现：

```
net.layer{i}.transferFcn = 'template_transfer'
```

传递函数的调用格式为：

A = template_transfer(***N***)

其中 ***N*** 为 $S \times Q$ 的网络输入向量。

函数返回值 ***A*** 为网络输出向量。在应用到网络上之后，生成的网络就会调用此传递函数进行计算与仿真。

神经网络工具箱提供了一个示例自定义函数，名为 mytf。输入命令 help mytf 可以查看此函数的相关信息；而输入命令 type mytf 则可以查看其源文件。

如同一般的 MATLAB 函数源文件一样，mytf 函数源文件前一部分为函数说明，包括函数自定义性质、函数调用格式、函数信息格式，以及一个简单示例。

第二部分为函数的具体实现，可以看到，当输入为字符串时，函数将输出自定义信息；而当输入为数字时，函数的结果由计算式 a = 1./(n.^8+1) 确定。

可以绘出此函数曲线，在 MATLAB 中输入如下命令：

```
n = -5:.1:5;
a = mytf(n);
plot(n,a)
```

绘出的曲线如图 19-6 所示。

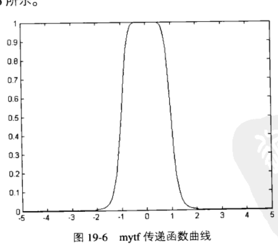


图 19-6　mytf 传递函数曲线

可以用函数 mytf 为模板来自定义传递函数。

2. 导数传递函数

所谓导数传递函数，是针对一个原传递函数来说的，也就是说，导数传递函数就是原传递函数的导数。

对于自定义传递函数 mytf，工具箱提供了它所对应的导数传递函数 mydtf。输入命令 type mydtf 可以查看其源文件。

同 mytf 函数源文件类似，mydtf 函数源文件前一部分为函数说明，第二部分为函数的具体实现。从源文件实现代码中可以看到，函数的返回值由计算式 d =-8*n.^7.*a.^2 确定。

同样，我们也可以绘出此导数传递函数的曲线。在 MATLAB 中输入如下命令：

```
n = -5:.1:5;
a = mytf(n);
da_dn = mydtf(n,a);
subplot(2,1,1), plot(n,a)
subplot(2,1,2), plot(n,da_dn)
```

绘出的曲线如图 19-7 所示。

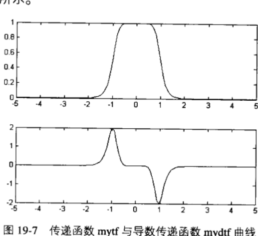


图 19-7 传递函数 mytf 与导数传递函数 mydtf 曲线

可以用函数 mydtf 为模板来自定义新的导数传递函数。

3. 网络输入函数

网络输入函数根据给定的加权输入向量 $\boldsymbol{Z}$，计算得到某层的网络输入向量 $\boldsymbol{N}$，网络输入向量与加权输入向量必须具有相同的维数。

网络输入函数经过定义之后，可以应用到任意的网络层上。例如自定义的网络输入函数为 tamplate_netinput，将其应用在网络的第 i 层上，可以通过下面的语句实现：

```
net.layer{i}.netInputFcn = 'template_netinput'
```

网络输入函数的调用格式为：

$\boldsymbol{N}$ = template_netinput(Z1, Z2…)

其中 Z1,Z2…为加权输入向量。函数返回值 $\boldsymbol{N}$ 为网络输入向量。

神经网络工具箱提供了网络输入函数的自定义样本函数 mynif。输入命令 type mynif 可以查看其源文件。

mynif 函数源文件的前一部分为函数说明，第二部分为函数的具体实现。从源文件实现代码中可以看到，mynif 函数的返回值，也就是自定义输入向量，是由输入参数矩阵元素倒数求和得到的。

下面举例说明 mynif 函数的应用。

【例 19-2】 给定加权输入向量，查看 mynif 的运行结果。

解： 定义加权输入向量。输入如下命令：

```
z1 = rand(4,5);
z2 = rand(4,5);
z3 = rand(4,5);
```

然后调用函数 mynif。输入命令：

```
n = mynif(z1,z2,z3)
```

输出结果为：

```
n = 0.1987    0.1190    0.2513    0.1618    0.2061
    0.0315    0.0735    0.1219    0.0335    0.1392
    0.0965    0.1393    0.0878    0.0501    0.2486
    0.2921    0.1688    0.0295    0.0974    0.0290
```

也就是计算得到的网络输入向量。可以用 mynif 为模板来自定义新的网络输入函数。

4. 导数网络输入函数

所谓导数网络输入函数，也是针对一个原函数来说的，也就是说，导数网络输入函数就是原网络输入函数的导数。

对于网络输入函数 mynif，工具箱提供了它所对应的导数网络输入函数 mydnif。输入 type mydnif 命令可以查看其源文件。

与前面类似，mydnif 函数源文件前一部分为函数说明，第二部分为函数的具体实现。从源文件实现代码中可以看到，mydnif 需要与 mynif 函数一起调用，mydnif 函数的返回值，也就是网络输入的导数，此导数是由计算式 d = n.^2 .* z.^2 得到的，而其中求导的自变量 z 正是 mynif 的输入变量。

下面举例说明 mydnif 函数的应用。

【例 19-3】 给定加权输入向量，查看 mydnif 的运行结果。

解： 定义加权输入向量，并调用函数 mynif 与 mydnif。输入命令：

```
z1 = rand(4,5);
z2 = rand(4,5);
z3 = rand(4,5);
n = mynif(z1,z2,z3)
dn_dz1 = mydnif(z1,n)
dn_dz2 = mydnif(z2,n)
dn_dz3 = mydnif(z3,n)
```

```
n = mynif(z1,z2,z3)
```

输出结果为：

```
n =0.0967    0.1212    0.0928    0.1337    0.1149
   0.1567    0.1612    0.1432    0.1542    0.1405
   0.1450    0.1439    0.0603    0.1148    0.0102
   0.1945    0.0993    0.0291    0.0850    0.1363
dn_dz1 =0.0053    0.0117    0.0002    0.0119    0.0005
        0.0016    0.0239    0.0014    0.0014    0.0012
        0.0054    0.0062    0.0026    0.0114    0.0000
        0.0185    0.0002    0.0001    0.0009    0.0042
dn_dz2 =0.0012    0.0124    0.0012    0.0050    0.0043
        0.0169    0.0021    0.0066    0.0144    0.0043
        0.0072    0.0119    0.0000    0.0115    0.0000
        0.0114    0.0056    0.0000    0.0001    0.0021
dn_dz3 =0.0002    0.0004    0.0041    0.0009    0.0038
        0.0155    0.0094    0.0115    0.0198    0.0196
        0.0020    0.0014    0.0007    0.0003    0.0000
        0.0106    0.0042    0.0000    0.0049    0.0036
```

其中，***n*** 是计算得到的网络输入向量，dn_dz1，dn_dz2，dn_dz3 分别是 ***n*** 对 z1、z2、z3 的导数。

可以用函数 mydnif 为模板来自定义新的导数网络输入函数。

5．权值函数

权值函数根据给定的输入向量 ***P***，以及权值矩阵 ***W***，计算得到一个加权的网络输入向量或矩阵 ***Z***。

权值函数经过定义之后，可以应用到任意的网络层上。例如自定义的网络输入函数为 tamplate_netwf，将其应用在网络的第 *i* 层上，可以通过下面的语句实现：

```
net.inputWeights{i}.weightFcn = 'template_netwf'
```

传递函数的调用格式为：

Z = template_netwf(***W***,***P***)

其中：

- ***W*** 为网络权值矩阵；
- ***P*** 为输入向量矩阵。

函数返回值 ***Z*** 为加权输入向量。

神经网络工具箱提供了权值函数的自定义样本函数 mywf。输入 type mywf 命令可以查看其源文件。

同样，mywf 函数源文件前一部分为函数说明，第二部分为函数的具体实现。从源文件实现代码中可以看到，mywf 返回值 ***Z*** 也就是网络的加权输入值，此网络加权输入是由计算式 ***Z*** = ***W****(***P***.^2) 得到的，其中 ***W*** 和 ***P*** 是函数的输入变量，分别是网络权值矩阵和输入向量。

下面举例说明 mywf 函数的应用。

【例 19-4】 给定加权输入向量，查看函数 mywf 的执行结果。

解： 定义随机输入向量与权值矩阵，并调用函数 mywf。输入命令：

```
w = rand(1,5);
p = rand(5,1);
z = mywf(w,p)
```

输出结果为：

```
z =    0.6636
```

即为得到的网络加权输入值。

可以用 mywf 为模板来自定义新的权值函数。

6. 导数权值函数

所谓导数权值函数，是针对一个原函数来说的，也就是说，导数权值函数就是原权值函数的导数。

对于权值函数 mywf，工具箱提供了它所对应的导数权值函数 mydwf。输入 type mydwf 命令可以查看其源文件。

mydwf 函数与 mywf 函数的关系与上面各个导数函数与原函数的关系一样，此处不再详细叙述。

下面举例说明 mydwf 函数的应用。

【例 19-5】 给定加权输入向量，查看函数 mydwf 的运行结果。

解： 定义加权输入向量，并调用函数 mywf 与 mydwf。输入命令：

```
w = rand(1,5);
p = rand(5,1);
z = mywf(w,p)
dz_dp = mydwf('p',w,p,z)
dz_dw = mydwf('w',w,p,z)
```

输出的结果为：

```
z =    0.8401
dz_dp =    0.1174    1.5832    0.2108    0.2901    0.0422
dz_dw =
    0.0185
    0.7557
    0.3361
    0.3023
    0.0210
```

其中 z 为加权输入向量，dz_dp、dz_dw 分别为加权输入向量对于输入向量和权值向量的导数。

以 mywf 为模板可以自定义新的导数权值函数。

19.3.2 初始化函数

初始化函数包括三类：网络初始化函数、层初始化函数、权值/偏差初始化函数。下面分别对其进行介绍。

1．网络初始化函数

网络初始化函数将网络所有的权值和偏差设定为一个特定值，作为网络的初始状态。网络初始化函数经过定义之后，可以应用到任意的网络上。例如自定义的网络初始化函数为 tamplate_nif，将其应用在网络 net 上，可以通过下面的语句实现：

```
net.initFcn = 'template_nif'
```

网络初始化函数的调用格式为：

net = template_nif(net)

其中 net 为初始化网络，通常既是函数输入，同时也设定为返回值。

2．层初始化函数

层初始化函数将给定网络某层的所有权值和偏差设定为一个特定值，作为训练的初始状态。层初始化函数经过定义之后，可以应用到网络的任意层。例如自定义的网络输入函数为 template_lif，将其应用在网络 net 的第 *i* 层上，可以通过下面的语句实现：

```
net.layers{i}.initFcn='template_lif'
```

其中：

- net 为初始化网络；
- *i* 为待初始化的层。

层初始化函数的调用格式为：

net = template_lif(net, *i*)

其中 net 为初始化网络，通常既是函数输入，同时也设为返回值。

3．权值/偏差初始化函数

权值/偏差初始化函数将给定网络的权值和偏差设定为特定值，作为训练的初始值。权值/偏差初始化函数经过定义之后，可以应用到网络的任意权值和偏差上。例如自定义的网络输入函数为 template_wbif，将其应用在网络 net 的第 *i* 层的权值和偏差上，可以通过下面的语句实现：

```
net.biase{i}.initFcn='template_wbif'
net.inputWeights{i}.initFcn='template_wbif'
```

其中：

- net 为初始化网络，通常既是函数输入，同时也设为返回值；
- *i* 为待初始化的层。

对于权值/偏差初始化函数，工具箱提供了一个示例自定义函数 mywbif，该函数的调

用格式为：

$$\text{function } w = \text{mywbif}(s,\text{pr})$$

其中：

- 输入变量 s 为网络神经元数目，
- pr 为输入向量取值范围。

函数返回值 w 为网络权值/偏差初始化值。

输入 type mywbif 命令可以查看 mywbif 函数的源文件。可以看到，mywbif 函数源文件前一部分为函数说明，第二部分为函数的具体实现。下面举例说明其应用。

【例 19-6】 对权值/偏差进行初始化，查看函数 mywbif 的运行结果。

解： 定义权值/偏差输入向量，并调用函数 mywbif。输入命令：

```
W = mywbif(4,[0 1; -2 2])
b = mywbif(4,[1 1])
```

得到的结果为：

```
W =
    0.0853    0.0402
    0.0622    0.0076
    0.0351    0.0240
    0.0513    0.0123
b =0.0184
    0.0240
    0.0417
    0.0050
```

以上完成了向量 $\boldsymbol{W}$ 和 $\boldsymbol{b}$ 的初始化。

以 mywbif 为模板可以自定义新的权值/偏差初始化函数。

19.3.3 学习函数

MATLAB 工具箱中可供自定义的学习函数包括四种类型：训练函数、自适应训练函数、性能函数，以及权值/偏差学习函数。下面对其分别介绍。

1．训练函数

训练函数用来对网络进行训练，它将输入向量循环应用于网络中，从而更新网络的权值和偏差，使其达到符合求解问题的状态。训练在达到最大迭代次数或最小梯度误差、训练精度时停止，当然以最小梯度误差方式停止时，有可能是网络落入了局部极小点。

训练函数经过定义之后，可以应用到网络上。例如自定义的训练函数为 template_tf，要将其应用在网络 net 的训练中，可以通过下面的语句实现：

```
net.trainFcn='template_tf'
```

其中 net 为待训练的网络。

训练函数的调用格式为：

```
[net,tr]=template_tf(net, Pd, Tl, Ai, Q, TS, VV, TV)
```

其中 net 为初始化网络，既是函数输入，也是函数执行返回值，tr 为训练后的网络状态。

2. 自适应训练函数

自适应训练函数与训练函数不同的一点在于，自适应训练函数在每个输入时间段内都需要对网络进行更新，并进行仿真。

自适应训练函数经过定义之后，可以应用到网络上。例如自定义的自适应训练函数为 template_atf，要将其应用在网络 net 的训练上，可以通过下面的语句实现：

```
net.adaptFcn='template_atf'
```

其中 net 为待训练的网络。

自适应训练函数的调用格式为：

[net,tr]=template_atf(net, Pd, Tl, Ai, Q, TS)

其中 net 为初始化网络，既是函数输入，也是函数执行返回值，tr 为训练后的网络状态。

3. 性能函数

性能函数用于设定网络训练时期望达到最优的函数指标。性能函数经过定义之后，可以应用到网络上。例如自定义的性能函数为 template_performf，要将其应用在网络 net 的训练上，可以通过下面的语句实现：

```
net.performFcn='template_performf'
```

其中 net 为待训练的网络。此后，任何时候对网络进行训练和调整的时候，都将采用此函数作为最优化的性能指标。

函数执行时需要提供性能参数，获取默认参数的格式如下：

```
LP = template_performf ('pdefaults')
```

自定义性能函数的调用格式为：

perf=template_performf(***E***, XX, PP)

其中：

- ***E*** 为期望值矩阵；
- XX 为网络权值和偏差矩阵；
- PP 为网络参数。

如果 ***E*** 是一个 cell 数组，则必须转换为矩阵形式：

```
E=cell2mat(E)
```

MATLAB 神经网络工具箱提供了一个示例自定义函数 mypf。输入 type mypf 命令可以查看其源文件。有兴趣的读者可以自行输入命令查看。

下面举例说明 mypf 函数的应用。

【例 19-7】 设定随机期望和权值偏差，查看函数 mypf 的运行结果。

解： 定义期望和权值偏差矩阵，并调用函数 mypf。输入命令：

```
e = rand(4,5);
x = rand(12,1);
pp = mypf('pdefaults')
perf = mypf(e,x,pp)
```

得到的结果为：

```
pp = x: 1
     y: 0.5000
perf = 0.7066
```

可以用函数 mypf 为模板来自定义新的性能函数。

4．权值/偏差学习函数

权值/偏差学习函数是一种比较特殊的学习函数，它需要和某些训练函数一起使用，才能在学习中实现对单独权值和偏差的更新。

权值/偏差学习函数经过定义之后，可以应用到网络上。例如自定义的权值/偏差学习函数为 template_wblf，要将其应用在网络 net 第 i 层偏差的训练中，可以通过下面的语句实现：

```
net.biases{i}.learnFcn='template_wblf'
```

其中 net 为待训练的网络。此后，任何时候对网络进行训练和调整的时候，都将采用此函数作为最优化的性能指标。

函数执行时需要提供学习函数参数，获取默认参数的格式如下：

```
LP = template_wblf ('pdefaults')
```

· 权值/偏差学习函数的调用格式为：

[db, LS]=template_wblf(***W***, ***P***, ***Z***, ***N***, ***A***, ***T***, ***E***, gW, gA, D, LP, LS)

其中：

- ***W*** 为 $S \times R$ 的权值矩阵；
- ***P*** 为 $R \times Q$ 的输入向量；
- ***Z*** 为 $S \times Q$ 的加权输入向量；
- ***N*** 为 $S \times Q$ 的网络输入向量；
- ***A*** 为 $S \times Q$ 的输出向量；
- ***T*** 为 $S \times Q$ 的层目标向量；
- ***E*** 为 $S \times Q$ 的层误差向量；
- gW 为 $S \times R$ 的性能相关梯度；
- gA 为 $S \times Q$ 的性能相关输出梯度；





- D 为 $S \times S$ 的神经元距离；
- LP 为学习参数；
- LS 为学习状态，初始值为[]。

函数的返回值中：

- db 为 $S \times R$ 的权值偏差调整矩阵；
- LS 为新的学习状态。

对于权值/偏差学习函数，工具箱提供了一个示例自定义函数 mywblf。输入 type mywblf 命令可以查看其源文件，其源程序不再详细列出，有兴趣的读者可以自行查看。

下面举例说明 mywblf 函数的应用。

【例 19-8】 设定随机期望和权值偏差，查看函数 mywblf 的运行结果。

解： 定义随机权值矩阵、性能相关梯度等。输入命令：

```
W = rand(4,5);
gW = rand(4,5);
```

然后获取默认参数。输入命令：

```
lp = mywblf('pdefaults')
```

输出结果为：

```
lp =  lr: 0.0100
```

调用 mywblf 函数，计算更新值。输入命令：

```
[dW,ls] = mywblf(w,[],[],[],[],[],[],gW,[],[],lp,[]);
W = W + dW,
gW = rand(4,5);
[dW,ls] = mywblf(w,[],[],[],[],[],[],gW,[],[],lp,ls);
W = W + dW,
```

输出结果为：

```
W =0.9292     0.2377     0.5226     0.6818     0.0378
   0.7311     0.4610     0.2339     0.3965     0.8874
   0.4896     0.9634     0.4910     0.3695     0.9148
   0.5806     0.5488     0.6268     0.9886     0.7976
W =0.9355     0.2433     0.5288     0.6887     0.0382
   0.7354     0.4650     0.2341     0.4015     0.8922
   0.4939     0.9647     0.4945     0.3730     0.9151
   0.5866     0.5504     0.6279     0.9919     0.7981
```

以 mywblf 为模板可以自定义新的权值/偏差初始化函数。

19.3.4 自组织映射函数

MATLAB 工具箱中可供自定义的自组织映射函数包括两种：拓扑函数和距离函数。下面对其分别介绍。

1．拓扑函数

拓扑函数用在自组织映射网络中，用来计算神经元的位置。拓扑函数在经过定义之后，可以应用到网络的任意一层。例如自定义的拓扑函数为 template_topf，要将其应用在网络 net 第 i 层上，可以通过下面的语句实现：

```
net.layers{i}.topologyFcn='template_topf'
```

拓扑函数的用途是从维数 dim 计算出神经元位置，其调用格式为：

pos = template_topf(dim1, dim2, … ,dimN)

其中，返回值 pos 是神经元的位置。

对于拓扑函数，MATLAB 工具箱提供了一个示例自定义函数 mytopf。输入 type mytopf 命令可以查看其源文件。函数源程序不再详细列出，有兴趣的读者可以自行输入命令查看。

下面举例说明 mytopf 函数的应用。

【例 19-9】 定义一个 20×20 的网络层，查看函数 mytopf 的运行结果。

解：输入命令：

```
pos = mytopf(20,20);
plotsom(pos)
```

输出的结果如图 19-8 所示。

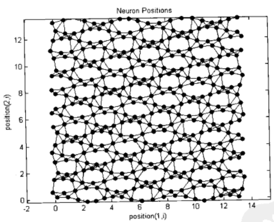


图 19-8 20×20 网络层拓扑结构

以函数 mytopf 为模板可以自定义新的拓扑函数。

2．距离函数

距离函数应用在自组织映射网络中，用于在给定位置的条件下，计算某一层神经元之间的距离。

距离函数在经过定义之后，可以应用到网络的任意一层。例如自定义的距离函数为

template_distf，要将其应用在网络 net 第 i 层，可以通过下面的语句实现：

```
net.layers{i}.distanceFcn='template_distf'
```

距离函数的用途是从位置 pos 计算出神经元间的距离，其调用格式为：

$$d = \text{template_distf}(\text{pos1, pos2}, \ldots ,\text{posN})$$

其中，返回值 d 是神经元的距离。

对于距离函数,MATLAB 工具箱提供了一个示例自定义函数 mydistf。输入 type mydistf 命令可以查看其源文件。函数源程序不再详细列出，有兴趣的读者可以自行输入命令查看。

下面对 mydistf 函数的应用进行举例说明。

【例 19-10】 定义一个网络拓扑结构，查看函数 mydistf 的运行结果。

解：输入命令：

```
pos = gridtop(3,2);
d = mydistf(pos)
```

输出的结果为：

```
d =      0    1.0000    2.0000    1.0000    1.5874    2.4473
    1.0000         0    1.0000    1.5874    1.0000    1.5874
    2.0000    1.0000         0    2.4473    1.5874    1.0000
    1.0000    1.5874    2.4473         0    1.0000    2.0000
    1.5874    1.0000    1.5874    1.0000         0    1.0000
    2.4473    1.5874    1.0000    2.0000    1.0000         0
```

返回结果为上面定义的网络拓扑层的距离矩阵。

以 mydistf 为模板可以自定义新的距离函数。

19.4 小结

本章讲述了 MATLAB 神经网络工具箱中一些用于自定义神经网络的函数，并通过实例讲述了这些函数的应用。熟练掌握自定义神经网络函数，有助于拓展 MATLAB 神经网络的应用。



附录 A 工具箱函数列表

1. 分析函数

函数名	说　明
errsurf	神经元误差面
confusion	混合矩阵分类
maxlinlr	线性神经元最大学习速率
roc	接收机运行特性

2. 距离函数

函数名	说　明
boxdist	两个位置向量之间的距离
dist	欧氏距离
linkdist	Link 距离
mandist	Manhattan 距离

3. 图形界面函数

函数名	说　明
nctool	神经网络分类工具
nftool	神经网络拟合工具
nntool	打开 Network/Manager 窗口
nntraintool	神经网络训练工具
nprtool	神经网络模式识别工具
view	查看神经网络

4. 层初始函数

函数名	说　明
initnw	Nguyen-Wdrow 层初始函数
initwb	权值偏差层初始函数

5. 学习函数

函数名	说　明
learncon	Consience 偏差学习函数
learngd	梯度下降学习函数
learngdm	带动量的梯度下降学习函数
learnh	Hebb 权重学习函数

续表

函 数 名	说　　明
learnhd	退化的 Hebb 学习函数
learnis	内星学习函数
learnk	Kohonen 权重学习函数
learnlv1	LVQ1 权重学习函数
learnlv2	LVQ2 权重学习函数
learnos	外星学习函数
learnp	感知器学习函数
learnpn	归一化感知器学习函数
learnsom	自组织映射权值学习函数
learnsomb	批处理自组织映射学习函数
learnwh	Widrow-Hoff 学习函数

6. 线搜索函数

函 数 名	说　　明
srchbac	回退搜索一维最小化
srchbre	Brent 一维间距定位
srchcha	Charalambous 一维最小化
srchgol	黄金分割一维最小化
rsrchhyb	二元/三元综合搜索的一维最小化

7. 网络输入函数

函 数 名	说　　明
netprod	网络输入求积
netsum	网络输入求和

8. 网络初始化函数

函 数 名	说　　明
initlay	逐层网络初始化

9. 网络应用函数

函 数 名	说　　明
adapt	允许网络自适应调节
disp	显示网络属性
display	显示网络变量名称和属性
init	初始化网络
sim	网络仿真
train	训练网络

10. 网络创建函数

函数名	说　明
network	自定义网络
newc	创建一个竞争层网络
newcf	创建一个级联前向网络
newdtdnn	创建一个时间分布延迟网络
newelm	创建一个 Elman 网络
newff	创建一个 BP 网络
newfftd	创建一个前向输入延迟 BP 网络
newfit	创建一个 Fitting 网络
newgrnn	创建一个 GRNN 网络
newhop	创建一个 Hopfield 网络
newlin	创建一个线性网络
newlind	创建一个线性层
newlrn	创建一个层反馈网络
newlvq	创建一个 LVQ 网络
newnarx	创建一个输出到输入有反馈的 BP 网络
newnarxsp	创建一个 NARX 网络
newp	创建一个感知器神经元
newpnn	创建一个概率神经网络
newpr	创建一个模式识别网络
newrb	创建一个径向基网络
newrbe	精确创建一个径向基网络
newsom	创建一个自组织映射网络
sp2narx	创建一个序列并行 NARX 网络

11. 性能函数

函数名	说　明
mae	平均绝对误差性能函数
mse	均方误差性能函数
msereg	均方误差 Regularization 性能函数
mseregec	均方误差 Regularization/Economization 性能函数
sse	误差平方和性能函数

12. 绘图函数

函数名	说　明
hintonw	权值矩阵 Hinton 图
hintonwb	权值矩阵和偏差向量 Hinton 图
plotbr	绘制应用贝叶斯规则训练的网络

续表

函 数 名	说　明
plotconfusion	绘制分类混合矩阵
plotep	绘制误差曲面上的权值偏差位置
plotes	绘制单输入神经元的误差曲面
plotfit	绘制函数拟合曲线
plotpc	绘制感知器向量图上的分界线
plotperf	绘制网络性能曲线
plotperform	绘制网络性能曲面
plotpv	绘制感知器输入目标向量
plotregression	绘制线性回归
plotroc	绘制接收器运行特性
plotsom	绘制自组织映射
plotsomhits	绘制自组织映射样本命中
plotsomnc	绘制自组织映射相邻连接
plotsomnd	绘制自组织映射相邻距离
plotsompos	绘制自组织映射权值位置
plotsomtop	绘制自组织映射拓扑
plottrainstate	绘制训练状态
plotv	绘制向量
plotvec	绘制不同颜色的向量
postreg	训练后网络后处理

13. 处理函数

函 数 名	说　明
fixunknowns	标记向量中含未知值的行
mapminmax	将向量行最大和最小值映射到[-1 1]
mapstd	将向量每行均值映射为 0 方差为 1
processpca	对每列进行主成分分析
removeconstantrows	删除常数列
removerows	删除特定行

14. Simulink 支持函数

函 数 名	说　明
gensim	生成 Simulink 模块

15. 拓扑函数

函 数 名	说　明
gridtop	矩形拓扑函数
hextop	六角形拓扑函数
randtop	随机拓扑函数

16. 训练函数

函数名	说明
trainb	权值偏差批处理训练
trainbfg	拟牛顿 BP 训练
trainbfgc	参考自适应控制器拟牛顿 BP 训练
trainbr	贝叶斯标准化训练
trainbuwb	批处理非监督权值偏差训练
trainc	循环递增修正训练
traincgb	Powell-Beale 共轭梯度的 BP 训练算法
traincgf	Fletcher-Powell 共轭梯度的 BP 训练算法
traincgp	Polak-Ribiere 共轭梯度的 BP 训练算法
traingd	梯度下降算法
traingda	自适应学习速率梯度下降算法
traingdm	带动量的梯度下降算法
traingdx	带动量的自适应学习速率梯度下降算法
trainlm	Levenberg-Marquartdt BP 训练算法
trainoss	一步正割 BP 算法
trainr	随即依次递增修正训练算法
trainrp	可复位的 BP 训练算法
trains	顺序依次递增训练算法
trainscg	量化连续梯度 BP 训练算法

17. 传递函数

函数名	说明
compet	竞争传递函数
hardlim	硬限值传递函数
hardlims	对称硬限值传递函数
logsig	logsig 传递函数
netinv	反向传递函数
poslin	正线性传递函数
purelin	线性传递函数
radbas	径向基传递函数
satlin	饱和线性传递函数
satlins	对称饱和线性传递函数
softmax	软最大值传递函数
tansig	tansig 传递函数
tribas	三角径向基传递函数

18. 有效函数

函数名	说明
calcgx	用单个向量计算权值和偏差性能梯度
calcjejj	计算雅克比性能向量

续表

函数名	说　明
calcjx	用单个向量计算权值和偏差性能梯度
calcpd	计算延时网络输入
calcperf	计算网络输出、信号和性能
getx	用单一向量获取所有网络权值和偏差
setx	用单一向量设置所有网络权值和偏差

19. 向量函数

函数名	说　明
combvec	创建所有向量组合
con2seq	将并行向量转为序列向量
concur	创建并行偏差向量
ind2vec	将标量转换为向量
minmax	矩阵行向量的范围
normc	矩阵列向量的归一化
normr	矩阵行向量的归一化
pnormc	矩阵列向量的拟归一化
quant	将数值离散为数量积
seq2con	将序列向量转为并行向量
vec2ind	将向量转为序号

20. 权值偏差初始函数

函数名	说　明
initcon	Conscience 偏差初始函数
initsompc	SOM 权值主成分初始化
initzero	零权值偏差初始函数
midpoint	中值初始函数
randnc	归一化列权值初始函数
randnr	归一化行权值初始函数
rands	对称随机权值偏差初始函数
revert	改变网络权值偏差至前一次的初始化函数

21. 权值函数

函数名	说　明
convwf	卷积权值函数
dist	欧氏距离权值函数
dotprod	点积权值函数
mandist	Manhattan 距离权值函数
negdist	负距离权值函数
normprod	归一化点积权值函数
scalprod	标量积权值函数

参考文献

[1] 丛爽. 面向 MATLAB 工具箱的神经网络理论和应用[M]. 合肥：中国科技大学出版社，1998.
[2] 戴小军. BP 神经网络用于市场预测的研究[D]. 武汉理工大学，2006.
[3] 董长虹. MATLAB 神经网络与应用[M]. 2 版. 北京：国防工业出版社，2007.
[4] 段军伟. 基于神经网络的绩优股票走势分析系统研究[D]. 燕山大学，2007.
[5] 方建卫. 人口预测方法的研究和改进[D]. 成都理工大学，2008.
[6] 飞思科技产品研发中心. MATLAB 6.5 辅助神经网络分析与设计[M]. 北京：电子工业出版社，2003.
[7] 冯剑，廖毅，王纪全. 水循环系统中基于神经网络的故障诊断[J]. 湘潭：湖南工程学院学报，2005.
[8] 葛哲学，孙志强. 神经网络理论与 MATLAB R2007 实现[M]. 北京：电子工业出版社，2008.
[9] 龚纯，王正林. MATLAB 常用算法程序集[M]. 北京：电子工业出版社，2008.
[10] 龚纯，王正林. 精通 MATLAB 最优化计算. 北京：电子工业出版社，2009
[11] 郭洁. 上市公司信用风险评价中的统计方法[D]. 北京交通大学，2009.
[12] 黄志丹. 基于神经网络的电信业务量预测[D]. 华中师范大学，2008.
[13] 黄加亮. RBF 神经网络在船用低速柴油机故障诊断中的应用研究[D]. 大连海事大学，2000.
[14] 靳蕃. 神经计算智能基础原理方法[M]. 成都：西南交通大学出版社，2000.
[15] 李洪东. 基于 BP 神经网络的汽车 ABS 系统故障诊断[D]. 吉林大学，2008.
[16] 潘林. 基于小波分析与神经网络的股票市场预测应用研究[D]. 武汉理工大学，2006.
[17] 汤建明. 基于神经网络的股市预测[D]. 华中科技大学，2006.
[18] 吴华星. 基于神经网络的股票价格预测[D]. 中国科学院，1998.
[19] 王正林，龚纯，何倩. 精通 MATLAB 科学计算. 2 版. 北京：电子工业出版社，2009
[20] 王正林，刘明. 精通 MATLAB 7[M]. 北京：电子工业出版社，2006.
[21] 杨高波，亓波. 精通 MATLAB 7.0 混合编程[M]. 北京：电子工业出版社，2006.
[22] 阎平凡，张长水. 人工神经网络与模拟进化计算[M]. 2 版. 北京：清华大学出版社，2006.